

fmtcount.sty: Displaying the Values of L^AT_EX Counters

Nicola L.C. Talbot

Vincent Belaïche

www.dickimaw-books.com

2015-05-05 (version 3.01)

Contents

1	Introduction	2
2	Available Commands	2
3	Package Options	8
4	Multilingual Support	8
4.1	Options for setting ordinal ending position raise/level	9
4.2	Options for French	10
4.3	Prefixes	15
5	Configuration File <code>fmtcount.cfg</code>	15
6	LaTeX2HTML style	15
7	Acknowledgements	16
8	Troubleshooting	16
9	The Code	16
9.0.1	<code>fc-american.def</code>	16
9.0.2	<code>fc-british.def</code>	17
9.0.3	<code>fc-english.def</code>	17
9.0.4	<code>fc-francais.def</code>	28
9.0.5	<code>fc-french.def</code>	28
9.0.6	<code>fc-frenchb.def</code>	58
9.0.7	<code>fc-german.def</code>	59
9.0.8	<code>fc-germanb.def</code>	69
9.0.9	<code>fc-italian</code>	69
9.0.10	<code>fc-ngerman.def</code>	70

9.0.11	<code>fc-ngermanb.def</code>	71
9.0.12	<code>fc-portuges.def</code>	71
9.0.13	<code>fc-portuguese.def</code>	86
9.0.14	<code>fc-spanish.def</code>	87
9.0.15	<code>fc-UKenglish.def</code>	104
9.0.16	<code>fc-USenglish.def</code>	105
9.1	<code>fnumparser.sty</code>	105
9.2	<code>fcprefix.sty</code>	116
9.3	<code>fmtcount.sty</code>	126
9.4	Multilingual Definitions	152

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: 3rd. 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 memoir'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: 3rd! whereas `\ordinal{section}[m] !` will produce: 3rd !

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: 3rd.

`\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 argument¹.

`\numberstringnum`

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

`\Numberstringnum`

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

`\NUMBERstringnum`

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

¹See all the various postings to `comp.text.tex` about `\MakeUppercase`

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

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

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

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

This does the same as `\ordinalstring`, but with initial letters in uppercase. For example, `\Ordinalstring{section}` will produce: 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{<label>}{<counter>}[<gender>]</code>
<code>\storeordinalstring</code>	<code>\storeordinalstring{<label>}{<counter>}[<gender>]</code>
<code>\storeOrdinalstring</code>	<code>\storeOrdinalstring{<label>}{<counter>}[<gender>]</code>
<code>\storeORDINALstring</code>	<code>\storeORDINALstring{<label>}{<counter>}[<gender>]</code>
<code>\storenumberstring</code>	<code>\storenumberstring{<label>}{<counter>}[<gender>]</code>
<code>\storeNumberstring</code>	<code>\storeNumberstring{<label>}{<counter>}[<gender>]</code>
<code>\storeNUMBERstring</code>	<code>\storeNUMBERstring{<label>}{<counter>}[<gender>]</code>
<code>\storeordinalnum</code>	<code>\storeordinalnum{<label>}{<number>}[<gender>]</code>
<code>\storeordinalstringnum</code>	<code>\storeordinalstring{<label>}{<number>}[<gender>]</code>
<code>\storeOrdinalstringnum</code>	<code>\storeOrdinalstringnum{<label>}{<number>}[<gender>]</code>
<code>\storeORDINALstringnum</code>	<code>\storeORDINALstringnum{<label>}{<number>}[<gender>]</code>
<code>\storenumberstringnum</code>	<code>\storenumberstring{<label>}{<number>}[<gender>]</code>

\storeNumberstringnum	<code>\storeNumberstring{<label>}{<number>} [<gender>]</code>
\storeNUMBERstringnum	<code>\storeNUMBERstring{<label>}{<number>} [<gender>]</code>
\binary	<code>\binary{<counter>}</code>
	This will print the value of <i><counter></i> as a binary number. E.g. <code>\binary{section}</code> will produce: 11. The declaration
\padzeroes	<code>\padzeroes [<n>]</code>
	will ensure numbers are written to <i><n></i> digits, padding with zeroes if necessary. E.g. <code>\padzeroes [8] \binary{section}</code> will produce: 00000011. The default value for <i><n></i> is 17.
\binarynum	<code>\binary{<n>}</code>
	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.
\octal	<code>\octal{<counter>}</code>
	This will print the value of <i><counter></i> 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.
\octalnum	<code>\octalnum{<n>}</code>
	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.
\hexadecimal	<code>\hexadecimal{<counter>}</code>
	This will print the value of <i><counter></i> 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.
\Hexadecimal	<code>\Hexadecimal{<counter>}</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:

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

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

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

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

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

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

4 Multilingual Support

Version 1.02 of the `fmtcount` package now has limited multilingual support. The following languages are implemented: English, Spanish, Portuguese, French, French (Swiss) and French (Belgian). German support was added in version 1.1.² Italian support was added in version 1.31.³

²Thanks to K. H. Fricke for supplying the information.

³Thanks to Edoardo Pasca for supplying the information.

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

`\FCloadlang`

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

in the preamble. The *<dialect>* should match the options passed to `babel` or `polyglossia`. `fmtcount` currently supports the following *<dialect>*: `english`, `UKenglish`, `british`, `USenglish`, `american`, `spanish`, `portuges`, `french`, `frenchb`, `francais`, `german`, `germanb`, `ngerman`, `ngermanb`, and `italian`. 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 § 4.2.

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

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

4.1 Options for setting ordinal ending position raise/level

`\fmtcountsetoptions`

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

where *<language>* is one of the supported language *<type>* is either `level` or `raise` or `undefine`. If the value is `level` or `raise`, then that will set the `fmtord` option accordingly⁴ only for that language *<language>*. If the value is `undefine`, then the non-language specific behaviour is followed.

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

⁴see § 3

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

4.2 Options for French

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

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

`\fmtcountsetoptions`

```
\fmtcountsetoptions{french={⟨french options⟩}}
```

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

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

`dialect`

```
\fmtcountsetoptions{french={dialect={⟨dialect⟩}}
```

`french`

```
\fmtcountsetoptions{french=⟨dialect⟩}
```

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

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

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

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

`swiss` septante pour 70, huitante⁵ pour 80, et nonante pour 90

Il est à noter que la variante `belgian` est parfaitement correcte pour les francophones français⁶, 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

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

⁶Je précise que l'auteur de ces lignes est français

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^e (par défaut), ou
false pour produire des ordinaux de la forme 2^{ème}

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

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

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

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

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

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

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

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

- la règle de l’accord en nombre des noms de nombre dans un numéral cardinal dépend de savoir s’il a vraiment une valeur cardinale ou bien une valeur ordinale, ainsi on écrit « aller à la page deux-cent (sans s) d’un livre de deux-cents (avec s) pages », il faut donc pouvoir changer la configuration pour sélectionner le cas considéré,

- un autre cas demandant quelque configurabilité est celui de « mil » et « mille ». Pour rappel « mille » est le pluriel irrégulier de « mil », mais l'alternance mil/mille est rare, voire pédante, car aujourd'hui « mille » n'est utilisé que comme un mot invariable, en effet le sort des pluriels étrangers est systématiquement de finir par disparaître comme par exemple « scénarii » aujourd'hui supplanté par « scénarios ». Pour continuer à pouvoir écrire « mil », il aurait fallu former le pluriel comme « mils », ce qui n'est pas l'usage. Certaines personnes utilisent toutefois encore « mil » dans les dates, par exemple « mil neuf cent quatre-vingt quatre » au lieu de « mille neuf cent quatre-vingt quatre »,
- finalement les règles du français quoique bien définies ne sont pas très cohérentes et il est donc inévitable qu'un jour ou l'autre on les simplifie. Le paquetage `fmtcount` est déjà prêt à cette éventualité.

Le paramètre *(french plural control)* peut prendre les valeurs suivantes :

<code>traditional</code>	pour sélectionner la règle en usage chez les adultes à la date de parution de ce document, et dans le cas des numéraux cardinaux, lorsqu'ils ont une valeur cardinale,
<code>reformed</code>	pour suivre toute nouvelle recommandation à la date de parution de ce document, , et dans le cas des numéraux cardinaux, lorsqu'ils ont une valeur cardinale, l'idée des options <code>traditional</code> et <code>reformed</code> est donc de pouvoir contenter à la fois les anciens et les modernes, mais à dire vrai à la date où ce document est écrit elles ont exactement le même effet,
<code>traditional o</code>	pareil que <code>traditional</code> mais dans le cas des numéraux cardinaux, lorsqu'ils ont une valeur ordinale,
<code>reformed o</code>	pareil que <code>reformed</code> mais dans le cas des numéraux cardinaux, lorsqu'ils ont une valeur ordinale, de même que précédemment <code>reformed o</code> et <code>traditional o</code> ont exactement le même effet,
<code>always</code>	pour marquer toujours le pluriel, ceci n'est correct que pour « mil » vis à vis des règles en vigueur,
<code>never</code>	pour ne jamais marquer le pluriel, ceci est incorrect vis à vis des règles d'orthographe en vigueur,
<code>multiple</code>	pour marquer le pluriel lorsque le nombre considéré est multiplié par au moins 2, ceci est la règle en vigueur pour les nombres de la forme $\langle n \rangle$ illion et $\langle n \rangle$ illiard lorsque le nombre a une valeur cardinale,

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 <n>illion ou un <n>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</i> ’est pas <i>globalement</i> en dernière position, où “globalement” a la même signification que pour l’option multiple g-last ; ceci est la règle que j’inferè être en vigueur pour les nombres de la forme <n>illion et <n>illiard lorsque le nombre a une valeur ordinale, mais à dire vrai pour des nombres aussi grands, par exemple « deux millions », je pense qu’il n’est tout simplement pas d’usage de dire « l’exemplaire deux million(s ?) » pour « le deux millionième exemplaire ».

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

<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 acti-

ver l’alternance mil/mille.

- `\fmtcountsetoptions{french={all plural=reformed}}` pour revenir dans la configuration par défaut.

dash or space

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

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

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

Par défaut, l’option vaut `reformed`.

scale

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

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

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

Par défaut, l’option vaut `recursive`.

n-illiard upto

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

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

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

mil plural mark

```
\fmtcountsetoptions{french={mil plural mark=\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).

4.3 Prefixes

latinnumeralstring

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

latinnumeralstringnum

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

5 Configuration File `fmtcount.cfg`

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

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

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

and if `fmtcount.cfg` has the line:

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

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

6 LaTeX2HTML style

The \LaTeX 2HTML style file `fmtcount.perl` is provided. The following limitations apply:

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

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

This, I agree, is an unpleasant cludge.

7 Acknowledgements

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

8 Troubleshooting

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

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

Local Variables: coding: utf-8 End:

9 The Code

9.0.1 fc-american.def

American English definitions

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

Loaded fc-USenglish.def if not already loaded

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

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

```
3 \global\let\@ordinalMamerican\@ordinalMUSenglish
4 \global\let\@ordinalFamerican\@ordinalMUSenglish
5 \global\let\@ordinalNamerican\@ordinalMUSenglish
6 \global\let\@numberstringMamerican\@numberstringMUSenglish
7 \global\let\@numberstringFamerican\@numberstringMUSenglish
8 \global\let\@numberstringNamerican\@numberstringMUSenglish
9 \global\let\@NumberstringMamerican\@NumberstringMUSenglish
10 \global\let\@NumberstringFamerican\@NumberstringMUSenglish
11 \global\let\@NumberstringNamerican\@NumberstringMUSenglish
12 \global\let\@ordinalstringMamerican\@ordinalstringMUSenglish
13 \global\let\@ordinalstringFamerican\@ordinalstringMUSenglish
14 \global\let\@ordinalstringNamerican\@ordinalstringMUSenglish
```



```

15 \global\let\@OrdinalstringMamerican\@OrdinalstringMUSenglish
16 \global\let\@OrdinalstringFamerican\@OrdinalstringMUSenglish
17 \global\let\@OrdinalstringNamerican\@OrdinalstringMUSenglish

```

9.0.2 fc-british.def

British definitions

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

Load fc-english.def, if not already loaded

```
19 \FCloadlang{english}%
```

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

```

20 \global\let\@ordinalMbritish\@ordinalMenglish
21 \global\let\@ordinalFbritish\@ordinalMenglish
22 \global\let\@ordinalNbritish\@ordinalMenglish
23 \global\let\@numberstringMbritish\@numberstringMenglish
24 \global\let\@numberstringFbritish\@numberstringMenglish
25 \global\let\@numberstringNbritish\@numberstringMenglish
26 \global\let\@NumberstringMbritish\@NumberstringMenglish
27 \global\let\@NumberstringFbritish\@NumberstringMenglish
28 \global\let\@NumberstringNbritish\@NumberstringMenglish
29 \global\let\@ordinalstringMbritish\@ordinalstringMenglish
30 \global\let\@ordinalstringFbritish\@ordinalstringMenglish
31 \global\let\@ordinalstringNbritish\@ordinalstringMenglish
32 \global\let\@OrdinalstringMbritish\@OrdinalstringMenglish
33 \global\let\@OrdinalstringFbritish\@OrdinalstringMenglish
34 \global\let\@OrdinalstringNbritish\@OrdinalstringMenglish

```

9.0.3 fc-english.def

English definitions

```
35 \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.

```

36 \newcommand*\@ordinalMenglish[2]{%
37 \def\@fc@ord{}%
38 \@orgargctr=#1\relax
39 \@ordinalctr=#1%
40 \@FCmodulo{\@ordinalctr}{100}%
41 \ifnum\@ordinalctr=11\relax
42   \def\@fc@ord{th}%
43 \else
44   \ifnum\@ordinalctr=12\relax
45     \def\@fc@ord{th}%
46   \else
47     \ifnum\@ordinalctr=13\relax

```

```

48     \def\@fc@ord{th}%
49     \else
50     \@FCmodulo{\@ordinalctr}{10}%
51     \ifcase\@ordinalctr
52     \def\@fc@ord{th}%      case 0
53     \or \def\@fc@ord{st}% case 1
54     \or \def\@fc@ord{nd}% case 2
55     \or \def\@fc@ord{rd}% case 3
56     \else
57     \def\@fc@ord{th}%      default case
58     \fi
59     \fi
60     \fi
61 \fi
62 \edef#2{\number#1\relax\noexpand\fmtord{\@fc@ord}}%
63 }%
64 \global\let\@ordinalMenglish\@ordinalMenglish

```

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

```

65 \global\let\@ordinalFenglish=\@ordinalMenglish
66 \global\let\@ordinalNenglish=\@ordinalMenglish

```

Define the macro that prints the value of a T_EX 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.

```

67 \newcommand*\@@unitstringenglish[1]{%
68   \ifcase#1\relax
69   zero%
70   \or one%
71   \or two%
72   \or three%
73   \or four%
74   \or five%
75   \or six%
76   \or seven%
77   \or eight%
78   \or nine%
79 \fi
80 }%
81 \global\let\@@unitstringenglish\@@unitstringenglish

```

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

```

82 \newcommand*\@@tenstringenglish[1]{%
83   \ifcase#1\relax
84   \or ten%
85   \or twenty%
86   \or thirty%
87   \or forty%
88   \or fifty%

```

```

89 \or sixty%
90 \or seventy%
91 \or eighty%
92 \or ninety%
93 \fi
94 }%
95 \global\let\@@tenstringenglish\@@tenstringenglish

```

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

```

96 \newcommand*\@@teenstringenglish[1]{%
97 \ifcase#1\relax
98 ten%
99 \or eleven%
100 \or twelve%
101 \or thirteen%
102 \or fourteen%
103 \or fifteen%
104 \or sixteen%
105 \or seventeen%
106 \or eighteen%
107 \or nineteen%
108 \fi
109 }%
110 \global\let\@@teenstringenglish\@@teenstringenglish

```

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

```

111 \newcommand*\@@Unitstringenglish[1]{%
112 \ifcase#1\relax
113 Zero%
114 \or One%
115 \or Two%
116 \or Three%
117 \or Four%
118 \or Five%
119 \or Six%
120 \or Seven%
121 \or Eight%
122 \or Nine%
123 \fi
124 }%
125 \global\let\@@Unitstringenglish\@@Unitstringenglish

```

The tens:

```

126 \newcommand*\@@Tenstringenglish[1]{%
127 \ifcase#1\relax
128 \or Ten%
129 \or Twenty%
130 \or Thirty%
131 \or Forty%
132 \or Fifty%
133 \or Sixty%

```

```

134 \or Seventy%
135 \or Eighty%
136 \or Ninety%
137 \fi
138 }%
139 \global\let\@@Tenstringenglish\@@Tenstringenglish

```

The teens:

```

140 \newcommand*\@@Teenstringenglish[1]{%
141 \ifcase#1\relax
142 Ten%
143 \or Eleven%
144 \or Twelve%
145 \or Thirteen%
146 \or Fourteen%
147 \or Fifteen%
148 \or Sixteen%
149 \or Seventeen%
150 \or Eighteen%
151 \or Nineteen%
152 \fi
153 }%
154 \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.)

```

155 \newcommand*\@@numberstringenglish[2]{%
156 \ifnum#1>99999
157 \PackageError{fmtcount}{Out of range}%
158 {This macro only works for values less than 100000}%
159 \else
160 \ifnum#1<0
161 \PackageError{fmtcount}{Negative numbers not permitted}%
162 {This macro does not work for negative numbers, however
163 you can try typing "minus" first, and then pass the modulus of
164 this number}%
165 \fi
166 \fi
167 \def#2{}%
168 \@strctr=#1\relax \divide\@strctr by 1000\relax
169 \ifnum\@strctr>9
170 \divide\@strctr by 10
171 \ifnum\@strctr>1\relax
172 \let\@@fc@numstr#2\relax
173 \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
174 \@strctr=#1 \divide\@strctr by 1000\relax
175 \@FCmodulo{\@strctr}{10}%
176 \ifnum\@strctr>0\relax

```

```

177     \let\@@fc@numstr#2\relax
178     \edef#2{\@@fc@numstr-\@unitstring{\@strctr}}%
179     \fi
180 \else
181     \@strctr=#1\relax
182     \divide\@strctr by 1000\relax
183     \@FCmodulo{\@strctr}{10}%
184     \let\@@fc@numstr#2\relax
185     \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
186     \fi
187     \let\@@fc@numstr#2\relax
188     \edef#2{\@@fc@numstr\ \@thousand}%
189 \else
190     \ifnum\@strctr>0\relax
191         \let\@@fc@numstr#2\relax
192         \edef#2{\@@fc@numstr\@unitstring{\@strctr}\ \@thousand}%
193     \fi
194 \fi
195 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
196 \divide\@strctr by 100
197 \ifnum\@strctr>0\relax
198     \ifnum#1>1000\relax
199         \let\@@fc@numstr#2\relax
200         \edef#2{\@@fc@numstr\ }%
201     \fi
202     \let\@@fc@numstr#2\relax
203     \edef#2{\@@fc@numstr\@unitstring{\@strctr}\ \@hundred}%
204 \fi
205 \@strctr=#1\relax \@FCmodulo{\@strctr}{100}%
206 \ifnum#1>100\relax
207     \ifnum\@strctr>0\relax
208         \let\@@fc@numstr#2\relax
209         \edef#2{\@@fc@numstr\ \@andname\ }%
210     \fi
211 \fi
212 \ifnum\@strctr>19\relax
213     \divide\@strctr by 10\relax
214     \let\@@fc@numstr#2\relax
215     \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
216     \@strctr=#1\relax \@FCmodulo{\@strctr}{10}%
217     \ifnum\@strctr>0\relax
218         \let\@@fc@numstr#2\relax
219         \edef#2{\@@fc@numstr-\@unitstring{\@strctr}}%
220     \fi
221 \else
222     \ifnum\@strctr<10\relax
223         \ifnum\@strctr=0\relax
224             \ifnum#1<100\relax
225                 \let\@@fc@numstr#2\relax

```

```

226         \edef#2{\@fc@numstr\@unitstring{\@strctr}}%
227     \fi
228 \else
229     \let\@fc@numstr#2\relax
230     \edef#2{\@fc@numstr\@unitstring{\@strctr}}%
231 \fi
232 \else
233     \FCmodulo{\@strctr}{10}%
234     \let\@fc@numstr#2\relax
235     \edef#2{\@fc@numstr\@teenstring{\@strctr}}%
236 \fi
237 \fi
238 }%
239 \global\let\@numberstringenglish\@numberstringenglish

```

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

```

240 \DeclareRobustCommand{\@numberstringMenglish}[2]{%
241     \let\@unitstring=\@unitstringenglish
242     \let\@teenstring=\@teenstringenglish
243     \let\@tenstring=\@tenstringenglish
244     \def\@hundred{hundred}\def\@thousand{thousand}%
245     \def\@andname{and}%
246     \@numberstringenglish{#1}{#2}%
247 }%
248 \global\let\@numberstringMenglish\@numberstringMenglish

```

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

```

249 \global\let\@numberstringFenglish=\@numberstringMenglish
250 \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.

```

251 \newcommand*\@NumberstringMenglish[2]{%
252     \let\@unitstring=\@Unitstringenglish
253     \let\@teenstring=\@Teenstringenglish
254     \let\@tenstring=\@Tenstringenglish
255     \def\@hundred{Hundred}\def\@thousand{Thousand}%
256     \def\@andname{and}%
257     \@numberstringenglish{#1}{#2}%
258 }%
259 \global\let\@NumberstringMenglish\@NumberstringMenglish

```

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

```

260 \global\let\@NumberstringFenglish=\@NumberstringMenglish
261 \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:

```

262 \newcommand*\@unitthstringenglish[1]{%

```

```

263 \ifcase#1\relax
264   zeroth%
265   \or first%
266   \or second%
267   \or third%
268   \or fourth%
269   \or fifth%
270   \or sixth%
271   \or seventh%
272   \or eighth%
273   \or ninth%
274   \fi
275 }%
276 \global\let\@@unitthstringenglish\@@unitthstringenglish

  Next the tens:
277 \newcommand*\@@tenthstringenglish[1]{%
278   \ifcase#1\relax
279     \or tenth%
280     \or twentieth%
281     \or thirtieth%
282     \or fortieth%
283     \or fiftieth%
284     \or sixtieth%
285     \or seventieth%
286     \or eightieth%
287     \or ninetieth%
288   \fi
289 }%
290 \global\let\@@tenthstringenglish\@@tenthstringenglish

  The teens:
291 \newcommand*\@@teenthstringenglish[1]{%
292   \ifcase#1\relax
293     tenth%
294     \or eleventh%
295     \or twelfth%
296     \or thirteenth%
297     \or fourteenth%
298     \or fifteenth%
299     \or sixteenth%
300     \or seventeenth%
301     \or eighteenth%
302     \or nineteenth%
303   \fi
304 }%
305 \global\let\@@teenthstringenglish\@@teenthstringenglish

  As before, but with the first letter in upper case. The units:
306 \newcommand*\@@Unitthstringenglish[1]{%
307   \ifcase#1\relax

```

```

308 Zeroth%
309 \or First%
310 \or Second%
311 \or Third%
312 \or Fourth%
313 \or Fifth%
314 \or Sixth%
315 \or Seventh%
316 \or Eighth%
317 \or Ninth%
318 \fi
319 }%
320 \global\let\@@Unitthstringenglish\@@Unitthstringenglish

```

The tens:

```

321 \newcommand*\@@Tenthstringenglish[1]{%
322 \ifcase#1\relax
323 \or Tenth%
324 \or Twentieth%
325 \or Thirtieth%
326 \or Fortieth%
327 \or Fiftieth%
328 \or Sixtieth%
329 \or Seventieth%
330 \or Eightieth%
331 \or Ninetieth%
332 \fi
333 }%
334 \global\let\@@Tenthstringenglish\@@Tenthstringenglish

```

The teens:

```

335 \newcommand*\@@Teenthstringenglish[1]{%
336 \ifcase#1\relax
337 Tenth%
338 \or Eleventh%
339 \or Twelfth%
340 \or Thirteenth%
341 \or Fourteenth%
342 \or Fifteenth%
343 \or Sixteenth%
344 \or Seventeenth%
345 \or Eighteenth%
346 \or Nineteenth%
347 \fi
348 }%
349 \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.


```

350 \newcommand*{\@@ordinalstringenglish[2]}{%
351 \@strctr=#1\relax
352 \ifnum#1>99999
353 \PackageError{fmtcount}{Out of range}%
354 {This macro only works for values less than 100000 (value given: \number\@strctr)}%
355 \else
356 \ifnum#1<0
357 \PackageError{fmtcount}{Negative numbers not permitted}%
358 {This macro does not work for negative numbers, however
359 you can try typing "minus" first, and then pass the modulus of
360 this number}%
361 \fi
362 \def#2{}%
363 \fi
364 \@strctr=#1\relax \divide\@strctr by 1000\relax
365 \ifnum\@strctr>9\relax

  #1 is greater or equal to 10000

366 \divide\@strctr by 10
367 \ifnum\@strctr>1\relax
368 \let\@@fc@ordstr#2\relax
369 \edef#2{\@@fc@ordstr\@tenstring{\@strctr}}%
370 \@strctr=#1\relax
371 \divide\@strctr by 1000\relax
372 \@FCmodulo{\@strctr}{10}%
373 \ifnum\@strctr>0\relax
374 \let\@@fc@ordstr#2\relax
375 \edef#2{\@@fc@ordstr-\@unitstring{\@strctr}}%
376 \fi
377 \else
378 \@strctr=#1\relax \divide\@strctr by 1000\relax
379 \@FCmodulo{\@strctr}{10}%
380 \let\@@fc@ordstr#2\relax
381 \edef#2{\@@fc@ordstr\@teenstring{\@strctr}}%
382 \fi
383 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
384 \ifnum\@strctr=0\relax
385 \let\@@fc@ordstr#2\relax
386 \edef#2{\@@fc@ordstr\ \@thousandth}%
387 \else
388 \let\@@fc@ordstr#2\relax
389 \edef#2{\@@fc@ordstr\ \@thousand}%
390 \fi
391 \else
392 \ifnum\@strctr>0\relax
393 \let\@@fc@ordstr#2\relax
394 \edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
395 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
396 \let\@@fc@ordstr#2\relax
397 \ifnum\@strctr=0\relax

```

```

398     \edef#2{\@fc@ordstr\ \@thousandth}%
399     \else
400     \edef#2{\@fc@ordstr\ \@thousand}%
401     \fi
402 \fi
403 \fi
404 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
405 \divide\@strctr by 100
406 \ifnum\@strctr>0\relax
407   \ifnum#1>1000\relax
408     \let\@fc@ordstr#2\relax
409     \edef#2{\@fc@ordstr\ }%
410     \fi
411     \let\@fc@ordstr#2\relax
412     \edef#2{\@fc@ordstr\@unitstring{\@strctr}}%
413     \@strctr=#1\relax \@FCmodulo{\@strctr}{100}%
414     \let\@fc@ordstr#2\relax
415     \ifnum\@strctr=0\relax
416       \edef#2{\@fc@ordstr\ \@hundredth}%
417       \else
418         \edef#2{\@fc@ordstr\ \@hundred}%
419         \fi
420 \fi
421 \@strctr=#1\relax \@FCmodulo{\@strctr}{100}%
422 \ifnum#1>100\relax
423   \ifnum\@strctr>0\relax
424     \let\@fc@ordstr#2\relax
425     \edef#2{\@fc@ordstr\ \@andname\ }%
426     \fi
427 \fi
428 \ifnum\@strctr>19\relax
429   \@tmpstrctr=\@strctr
430   \divide\@strctr by 10\relax
431   \@FCmodulo{\@tmpstrctr}{10}%
432   \let\@fc@ordstr#2\relax
433   \ifnum\@tmpstrctr=0\relax
434     \edef#2{\@fc@ordstr\@tenthstring{\@strctr}}%
435     \else
436       \edef#2{\@fc@ordstr\@tenstring{\@strctr}}%
437       \fi
438   \@strctr=#1\relax \@FCmodulo{\@strctr}{10}%
439   \ifnum\@strctr>0\relax
440     \let\@fc@ordstr#2\relax
441     \edef#2{\@fc@ordstr-\@unitthstring{\@strctr}}%
442     \fi
443 \else
444   \ifnum\@strctr<10\relax
445     \ifnum\@strctr=0\relax
446       \ifnum#1<100\relax

```

```

447     \let\@@fc@ordstr#2\relax
448     \edef#2{\@@fc@ordstr\@unitthstring{\@strctr}}%
449     \fi
450   \else
451     \let\@@fc@ordstr#2\relax
452     \edef#2{\@@fc@ordstr\@unitthstring{\@strctr}}%
453     \fi
454   \else
455     \@FCmodulo{\@strctr}{10}%
456     \let\@@fc@ordstr#2\relax
457     \edef#2{\@@fc@ordstr\@teenthstring{\@strctr}}%
458   \fi
459 \fi
460 }%
461 \global\let\@@ordinalstringenglish\@ordinalstringenglish

```

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

```

462 \DeclareRobustCommand{\@ordinalstringMenglish}[2]{%
463   \let\@unitthstring=\@unitthstringenglish
464   \let\@teenthstring=\@teenthstringenglish
465   \let\@tenthstring=\@tenthstringenglish
466   \let\@unitstring=\@unitstringenglish
467   \let\@teenstring=\@teenstringenglish
468   \let\@tenstring=\@tenstringenglish
469   \def\@andname{and}%
470   \def\@hundred{hundred}\def\@thousand{thousand}%
471   \def\@hundredth{hundredth}\def\@thousandth{thousandth}%
472   \@ordinalstringenglish{#1}{#2}%
473 }%
474 \global\let\@ordinalstringMenglish\@ordinalstringMenglish

```

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

```

475 \global\let\@ordinalstringFenglish=\@ordinalstringMenglish
476 \global\let\@ordinalstringNenglish=\@ordinalstringMenglish

```

First letter of each word in upper case:

```

477 \DeclareRobustCommand{\@OrdinalstringMenglish}[2]{%
478   \let\@unitthstring=\@Unitthstringenglish
479   \let\@teenthstring=\@Teenthstringenglish
480   \let\@tenthstring=\@Tenthstringenglish
481   \let\@unitstring=\@Unitstringenglish
482   \let\@teenstring=\@Teenstringenglish
483   \let\@tenstring=\@Tenstringenglish
484   \def\@andname{and}%
485   \def\@hundred{Hundred}\def\@thousand{Thousand}%
486   \def\@hundredth{Hundredth}\def\@thousandth{Thousandth}%
487   \@ordinalstringenglish{#1}{#2}%
488 }%
489 \global\let\@OrdinalstringMenglish\@OrdinalstringMenglish

```

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

```
490 \global\let\@OrdinalstringFenglish=\@OrdinalstringMenglish
491 \global\let\@OrdinalstringNenglish=\@OrdinalstringMenglish
```

9.0.4 fc-francais.def

```
492 \ProvidesFCLanguage{francais}[2013/08/17]%
493 \FCloadlang{french}%
```

Set francais to be equivalent to french.

```
494 \global\let\@ordinalMfrancais=\@ordinalMfrench
495 \global\let\@ordinalFfrancais=\@ordinalFfrench
496 \global\let\@ordinalNfrancais=\@ordinalNfrench
497 \global\let\@numberstringMfrancais=\@numberstringMfrench
498 \global\let\@numberstringFfrancais=\@numberstringFfrench
499 \global\let\@numberstringNfrancais=\@numberstringNfrench
500 \global\let\@NumberstringMfrancais=\@NumberstringMfrench
501 \global\let\@NumberstringFfrancais=\@NumberstringFfrench
502 \global\let\@NumberstringNfrancais=\@NumberstringNfrench
503 \global\let\@ordinalstringMfrancais=\@ordinalstringMfrench
504 \global\let\@ordinalstringFfrancais=\@ordinalstringFfrench
505 \global\let\@ordinalstringNfrancais=\@ordinalstringNfrench
506 \global\let\@OrdinalstringMfrancais=\@OrdinalstringMfrench
507 \global\let\@OrdinalstringFfrancais=\@OrdinalstringFfrench
508 \global\let\@OrdinalstringNfrancais=\@OrdinalstringNfrench
```

9.0.5 fc-french.def

Definitions for French.

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

Package fcprefix is needed to format the prefix $\langle n \rangle$ in $\langle n \rangle$ illion or $\langle n \rangle$ illiard. Big

numbers were developed based on reference: 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'.
510 \def\fc@french@set@plural#1#2#3#4#5#6{%
511   \ifthenelse{\equal{#2}{reformed}}{%
512     \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{#3}%
513   }{%
514     \ifthenelse{\equal{#2}{traditional}}{%
515       \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{#4}%
```

```

516 }{%
517   \ifthenelse{\equal{#2}{reformed o}}{%
518     \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{#5}%
519   }{%
520     \ifthenelse{\equal{#2}{traditional o}}{%
521       \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{#6}%
522     }{%
523       \ifthenelse{\equal{#2}{always}}{%
524         \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{0}%
525       }{%
526         \ifthenelse{\equal{#2}{never}}{%
527           \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{1}%
528         }{%
529           \ifthenelse{\equal{#2}{multiple}}{%
530             \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{2}%
531           }{%
532             \ifthenelse{\equal{#2}{multiple g-last}}{%
533               \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{3}%
534             }{%
535               \ifthenelse{\equal{#2}{multiple l-last}}{%
536                 \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{4}%
537               }{%
538                 \ifthenelse{\equal{#2}{multiple lng-last}}{%
539                   \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{5}%
540                 }{%
541                   \ifthenelse{\equal{#2}{multiple ng-last}}{%
542                     \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{6}%
543                   }{%
544                     \PackageError{fmtcount}{Unexpected argument}{%
545                       '#2' was unexpected: french option '#1 plural' expects 'reformed'
546                       'reformed o', 'traditional o', 'always', 'never', 'multiple', 'mu
547                       'multiple l-last', 'multiple lng-last', or 'multiple ng-last'.%
548                     }
549 }}}}}}}}}}}}}

```

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'.

```

549 \def\@tempa#1#2#3{%
550   \define@key{fcfrench}{#1 plural}[reformed]{%
551     \fc@french@set@plural{#1}{##1}{#2}{#2}{#3}{#3}%
552   }%
553 }
554 \@tempa{vingt}{4}{5}
555 \@tempa{cent}{4}{5}
556 \@tempa{mil}{0}{0}
557 \@tempa{n-illion}{2}{6}
558 \@tempa{n-illiard}{2}{6}

```

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

```

559 \define@key{fcfrench}{all plural}[reformed]{%
560   \csname KV@fcfrench@vingt plural\endcsname{#1}%
561   \csname KV@fcfrench@cent plural\endcsname{#1}%
562   \csname KV@fcfrench@mil plural\endcsname{#1}%
563   \csname KV@fcfrench@n-illion plural\endcsname{#1}%
564   \csname KV@fcfrench@n-illiard plural\endcsname{#1}%
565 }

```

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

```

566 \define@key{fcfrench}{dash or space}[reformed]{%
567   \ifthenelse{\equal{#1}{traditional}}{%
568     \let\fc@frenchoptions@supermillion@dos\space%
569     \let\fc@frenchoptions@submillion@dos\space
570   }{%
571     \ifthenelse{\equal{#1}{reformed}\or\equal{#1}{1990}}{%
572       \let\fc@frenchoptions@supermillion@dos\space
573       \def\fc@frenchoptions@submillion@dos{-}%
574     }{%
575       \ifthenelse{\equal{#1}{always}}{%
576         \def\fc@frenchoptions@supermillion@dos{-}%
577         \def\fc@frenchoptions@submillion@dos{-}%
578       }{%
579         \PackageError{fmtcount}{Unexpected argument}{%
580           French option ‘dash or space’ expects ‘always’, ‘reformed’ or ‘traditional’
581         }
582       }%
583     }%
584   }%
585 }

```

Option ‘scale’, can take 3 possible values:

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

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

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

```

586 \define@key{fcfrench}{scale}[recursive]{%
587   \ifthenelse{\equal{#1}{long}}{%
588     \let\fc@poweroften\fc@pot@longscalefrench
589   }{%
590     \ifthenelse{\equal{#1}{recursive}}{%
591       \let\fc@poweroften\fc@pot@recursivefrench
592     }{%

```

```

593     \ifthenelse{\equal{#1}{short}}{%
594         \let\fc@poweroften\fc@pot@shortscalefrench
595     }{%
596         \PackageError{fmtcount}{Unexpected argument}{%
597             French option ‘scale’ expects ‘long’, ‘recursive’ or ‘short’
598         }
599     }%
600 }%
601 }%
602 }

```

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”

```

603 \define@key{fcfrench}{n-illiard upto}[infinity]{%
604     \ifthenelse{\equal{#1}{infinity}}{%
605         \def\fc@longscale@nilliard@upto{0}%
606     }{%
607         \ifthenelse{\equal{#1}{infty}}{%
608             \def\fc@longscale@nilliard@upto{0}%
609         }{%
610             \if Q\ifnum9<1#1Q\fi\else
611                 \PackageError{fmtcount}{Unexpected argument}{%
612                     French option ‘milliard threshold’ expects ‘infinity’, or equivalently ‘infty’, or
613                     integer.}%
614             \fi
615             \def\fc@longscale@nilliard@upto{#1}%
616         }%
617 }

```

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.

```

618 \def\@tempa#1{%
619     \define@key{fcfrench}{#1}[]{%
620         \PackageError{fmtcount}{Unexpected argument}{French option with key ‘#1’ does not take
621             any value}}%
622     \expandafter\def\csname KV@fcfrench@#1@default\endcsname{%
623         \def\fmtcount@french{#1}}%
624 }%
625 \@tempa{france}\@tempa{swiss}\@tempa{belgian}%

```

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

```

626 \def\fmtcount@french{france}%

```

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

cise.

```
627 \define@key{french}{dialect}[france]{%
628   \ifthenelse{\equal{#1}{france}
629     \or\equal{#1}{swiss}
630     \or\equal{#1}{belgian}}{%
631     \def\fmtcount@french{#1}}{%
632     \PackageError{fmtcount}{Invalid value ‘#1’ to french option dialect key}
633     {Option ‘french’ can only take the values ‘france’,
634       ‘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’`.

```
635 \define@key{french}{mil plural mark}[le]{%
636   \def\fc@frenchoptions@mil@plural@mark{#1}}
```

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

```
637 \def\fc@UpperCaseFirstLetter#1#2\@nil{%
638   \uppercase{#1}#2}
639
640 \def\fc@CaseIden#1\@nil{%
641   #1%
642 }
643 \def\fc@UpperCaseAll#1\@nil{%
644   \uppercase{#1}%
645 }
646
647 \let\fc@case\fc@CaseIden
648
```

`\@ordinalMfrench`

```
649 \newcommand*{\@ordinalMfrench}[2]{%
650 \iffmtord@abbrv
651   \ifnum#1=1 %
652     \edef#2{\number#1\relax\noexpand\fmtord{er}}%
653   \else
654     \edef#2{\number#1\relax\noexpand\fmtord{e}}%
655   \fi
656 \else
657   \PackageWarning{fmtcount}{Non abbreviated ordinal finals (‘eme’) are
658     considered incorrect in French.}%
659   \ifnum#1=1 %
660     \edef#2{\number#1\relax\noexpand\fmtord{er}}%
661   \else
662     \protected@edef#2{\number#1\relax\noexpand\fmtord{\protect\‘eme}}%
663   \fi
664 \fi}
```

`\@ordinalFfrench`

```
665 \newcommand*{\@ordinalFfrench}[2]{%
666 \iffmtord@abbrv
```



```

667 \ifnum#1=1 %
668   \edef#2{\number#1\relax\noexpand\fmtord{re}}%
669 \else
670   \edef#2{\number#1\relax\noexpand\fmtord{e}}%
671 \fi
672 \else
673 \PackageWarning{fmtcount}{Non abbreviated ordinal finals ('eme) are
674   considered incorrect in French.}%
675 \ifnum#1=1 %
676   \protected@edef#2{\number#1\relax\noexpand\fmtord{\protect\'ere}}%
677 \else
678   \protected@edef#2{\number#1\relax\noexpand\fmtord{\protect\'eme}}%
679 \fi
680 \fi}

```

In French neutral gender and masculine gender are formally identical.

```
681 \let\@ordinalNfrench\@ordinalMfrench
```

@@unitstringfrench

```

682 \newcommand*{\@unitstringfrench}[1]{%
683 \noexpand\fc@case
684 \ifcase#1 %
685 z\'ero%
686 \or un%
687 \or deux%
688 \or trois%
689 \or quatre%
690 \or cinq%
691 \or six%
692 \or sept%
693 \or huit%
694 \or neuf%
695 \fi
696 \noexpand\@nil
697 }

```

\@tenstringfrench

```

698 \newcommand*{\@tenstringfrench}[1]{%
699 \noexpand\fc@case
700 \ifcase#1 %
701 \or dix%
702 \or vingt%
703 \or trente%
704 \or quarante%
705 \or cinquante%
706 \or soixante%
707 \or septante%
708 \or huitante%
709 \or nonante%
710 \or cent%
711 \fi

```

```
712 \noexpand\@nil
713 }
```

`\@teenstringfrench`

```
714 \newcommand*\@teenstringfrench}[1]{%
715 \noexpand\fc@case
716 \ifcase#1 %
717     dix%
718 \or onze%
719 \or douze%
720 \or treize%
721 \or quatorze%
722 \or quinze%
723 \or seize%
724 \or dix\noexpand\@nil-\noexpand\fc@case sept%
725 \or dix\noexpand\@nil-\noexpand\fc@case huit%
726 \or dix\noexpand\@nil-\noexpand\fc@case neuf%
727 \fi
728 \noexpand\@nil
729 }
```

`\@seventiesfrench`

```
730 \newcommand*\@seventiesfrench}[1]{%
731 \@tenstring{6}%
732 \ifnum#1=1 %
733 \fc@frenchoptions@submillion@dos\@andname\fc@frenchoptions@submillion@dos
734 \else
735 -%
736 \fi
737 \@teenstring{#1}%
738 }
```

`\@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 “ $\langle n \rangle$ illion| $\langle n \rangle$ illiard”.

```
739 \newcommand*\@eightiesfrench[1]{%
740 \fc@case quatre\@nil-\noexpand\fc@case vingt%
741 \ifnum#1>0 %
742     \ifnum\fc@frenchoptions@vingt@plural=0 % vingt plural=always
743     s%
```

```

744 \fi
745 \noexpand\@nil
746 -\@unitstring{#1}%
747 \else
748 \ifcase\fc@frenchoptions@vingt@plural\space
749   s% 0: always
750 \or
751   % 1: never
752 \or
753   s% 2: multiple
754 \or
755   % 3: multiple g-last
756   \ifnum\count0=\count6\ifnum\count9=0 s\fi\fi
757 \or
758   % 4: multiple l-last
759   \ifnum\count9=1 %
760   \else
761   s%
762   \fi
763 \or
764   % 5: multiple lng-last
765   \ifnum\count9=1 %
766   \else
767   \ifnum\count0>0 %
768   s%
769   \fi
770   \fi
771 \or
772   % or 6: multiple ng-last
773   \ifnum\count0>0 %
774   s%
775   \fi
776 \fi
777 \noexpand\@nil
778 \fi
779 }
780 \newcommand*\@@@ninetiesfrench}[1]{%
781 \fc@case quatre\@nil-\noexpand\fc@case vingt%
782 \ifnum\fc@frenchoptions@vingt@plural=0 % vingt plural=always
783 s%
784 \fi
785 \noexpand\@nil
786 -\@teenstring{#1}%
787 }
788 \newcommand*\@@@seventiesfrenchswiss}[1]{%
789 \@tenstring{7}%
790 \ifnum#1=1\ \@andname\ \fi
791 \ifnum#1>1-\fi
792 \ifnum#1>0 \@unitstring{#1}\fi

```

```

793 }
794 \newcommand*{\@@eightiesfrenchswiss}[1]{%
795 \@tenstring{8}%
796 \ifnum#1=1\ \@andname\ \fi
797 \ifnum#1>1-\fi
798 \ifnum#1>0 \@unitstring{#1}\fi
799 }
800 \newcommand*{\@@ninetiesfrenchswiss}[1]{%
801 \@tenstring{9}%
802 \ifnum#1=1\ \@andname\ \fi
803 \ifnum#1>1-\fi
804 \ifnum#1>0 \@unitstring{#1}\fi
805 }

```

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

```

806 \newcommand*\fc@french@common{%
807 \let\@unitstring=\@unitstringfrench
808 \let\@teenstring=\@teenstringfrench
809 \let\@tenstring=\@tenstringfrench
810 \def\@hundred{cent}%
811 \def\@andname{et}%
812 }

813 \DeclareRobustCommand{\@numberstringMfrenchswiss}[2]{%
814 \let\fc@case\fc@CaseIden
815 \fc@french@common
816 \let\@seventies=\@seventiesfrenchswiss
817 \let\@eighties=\@eightiesfrenchswiss
818 \let\@nineties=\@ninetiesfrenchswiss
819 \let\fc@nbrstr@preamble\@empty
820 \let\fc@nbrstr@postamble\@empty
821 \@@numberstringfrench{#1}{#2}}
822 \DeclareRobustCommand{\@numberstringMfrenchfrance}[2]{%
823 \let\fc@case\fc@CaseIden
824 \fc@french@common
825 \let\@seventies=\@seventiesfrench
826 \let\@eighties=\@eightiesfrench
827 \let\@nineties=\@ninetiesfrench
828 \let\fc@nbrstr@preamble\@empty
829 \let\fc@nbrstr@postamble\@empty
830 \@@numberstringfrench{#1}{#2}}
831 \DeclareRobustCommand{\@numberstringMfrenchbelgian}[2]{%
832 \let\fc@case\fc@CaseIden
833 \fc@french@common
834 \let\@seventies=\@seventiesfrenchswiss
835 \let\@eighties=\@eightiesfrench
836 \let\@nineties=\@ninetiesfrench
837 \let\fc@nbrstr@preamble\@empty
838 \let\fc@nbrstr@postamble\@empty

```

```

839 \@@numberstringfrench{#1}{#2}}
840 \let\@numberstringMfrench=\@numberstringMfrenchfrance
841 \DeclareRobustCommand{\@numberstringFfrenchswiss}[2]{%
842 \let\fc@case\fc@CaseIden
843 \fc@french@common
844 \let\@seventies=\@@seventiesfrenchswiss
845 \let\@eighties=\@@eightiesfrenchswiss
846 \let\@nineties=\@@ninetiesfrenchswiss
847 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
848 \let\fc@nbrstr@postamble\@empty
849 \@@numberstringfrench{#1}{#2}}
850 \DeclareRobustCommand{\@numberstringFfrenchfrance}[2]{%
851 \let\fc@case\fc@CaseIden
852 \fc@french@common
853 \let\@seventies=\@@seventiesfrench
854 \let\@eighties=\@@eightiesfrench
855 \let\@nineties=\@@ninetiesfrench
856 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
857 \let\fc@nbrstr@postamble\@empty
858 \@@numberstringfrench{#1}{#2}}
859 \DeclareRobustCommand{\@numberstringFfrenchbelgian}[2]{%
860 \let\fc@case\fc@CaseIden
861 \fc@french@common
862 \let\@seventies=\@@seventiesfrenchswiss
863 \let\@eighties=\@@eightiesfrench
864 \let\@nineties=\@@ninetiesfrench
865 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
866 \let\fc@nbrstr@postamble\@empty
867 \@@numberstringfrench{#1}{#2}}
868 \let\@numberstringFfrench=\@numberstringFfrenchfrance
869 \let\@ordinalstringNfrench\@ordinalstringMfrench
870 \DeclareRobustCommand{\@NumberstringMfrenchswiss}[2]{%
871 \let\fc@case\fc@UpperCaseFirstLetter
872 \fc@french@common
873 \let\@seventies=\@@seventiesfrenchswiss
874 \let\@eighties=\@@eightiesfrenchswiss
875 \let\@nineties=\@@ninetiesfrenchswiss
876 \let\fc@nbrstr@preamble\@empty
877 \let\fc@nbrstr@postamble\@empty
878 \@@numberstringfrench{#1}{#2}}
879 \DeclareRobustCommand{\@NumberstringMfrenchfrance}[2]{%
880 \let\fc@case\fc@UpperCaseFirstLetter
881 \fc@french@common
882 \let\@seventies=\@@seventiesfrench
883 \let\@eighties=\@@eightiesfrench
884 \let\@nineties=\@@ninetiesfrench
885 \let\fc@nbrstr@preamble\@empty
886 \let\fc@nbrstr@postamble\@empty
887 \@@numberstringfrench{#1}{#2}}

```

```

888 \DeclareRobustCommand{\@NumberstringMfrenchbelgian}[2]{%
889 \let\fc@case\fc@UpperCaseFirstLetter
890 \fc@french@common
891 \let\@seventies=\@seventiesfrenchswiss
892 \let\@eighties=\@eightiesfrench
893 \let\@nineties=\@ninetiesfrench
894 \let\fc@nbrstr@preamble\@empty
895 \let\fc@nbrstr@postamble\@empty
896 \@@numberstringfrench{#1}{#2}}
897 \let\@NumberstringMfrench=\@NumberstringMfrenchfrance
898 \DeclareRobustCommand{\@NumberstringFfrenchswiss}[2]{%
899 \let\fc@case\fc@UpperCaseFirstLetter
900 \fc@french@common
901 \let\@seventies=\@seventiesfrenchswiss
902 \let\@eighties=\@eightiesfrenchswiss
903 \let\@nineties=\@ninetiesfrenchswiss
904 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
905 \let\fc@nbrstr@postamble\@empty
906 \@@numberstringfrench{#1}{#2}}
907 \DeclareRobustCommand{\@NumberstringFfrenchfrance}[2]{%
908 \let\fc@case\fc@UpperCaseFirstLetter
909 \fc@french@common
910 \let\@seventies=\@seventiesfrench
911 \let\@eighties=\@eightiesfrench
912 \let\@nineties=\@ninetiesfrench
913 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
914 \let\fc@nbrstr@postamble\@empty
915 \@@numberstringfrench{#1}{#2}}
916 \DeclareRobustCommand{\@NumberstringFfrenchbelgian}[2]{%
917 \let\fc@case\fc@UpperCaseFirstLetter
918 \fc@french@common
919 \let\@seventies=\@seventiesfrenchswiss
920 \let\@eighties=\@eightiesfrench
921 \let\@nineties=\@ninetiesfrench
922 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
923 \let\fc@nbrstr@postamble\@empty
924 \@@numberstringfrench{#1}{#2}}
925 \let\@NumberstringFfrench=\@NumberstringFfrenchfrance
926 \let\@NumberstringNfrench\@NumberstringMfrench
927 \DeclareRobustCommand{\@ordinalstringMfrenchswiss}[2]{%
928 \let\fc@case\fc@CaseIden
929 \let\fc@first=\fc@@firstfrench
930 \fc@french@common
931 \let\@seventies=\@seventiesfrenchswiss
932 \let\@eighties=\@eightiesfrenchswiss
933 \let\@nineties=\@ninetiesfrenchswiss
934 \@@ordinalstringfrench{#1}{#2}%
935 }
936 \newcommand*\fc@@firstfrench{premier}

```

```

937 \newcommand*\fc@@firstFfrench{premi\protect\‘ere}
938 \DeclareRobustCommand{\@ordinalstringMfrenchfrance}[2]{%
939 \let\fc@case\fc@CaseIden
940 \let\fc@first=\fc@@firstfrench
941 \fc@french@common
942 \let\@seventies=\@@seventiesfrench
943 \let\@eighties=\@@eightiesfrench
944 \let\@nineties=\@@ninetiesfrench
945 \@@ordinalstringfrench{#1}{#2}}
946 \DeclareRobustCommand{\@ordinalstringMfrenchbelgian}[2]{%
947 \let\fc@case\fc@CaseIden
948 \let\fc@first=\fc@@firstfrench
949 \fc@french@common
950 \let\@seventies=\@@seventiesfrench
951 \let\@eighties=\@@eightiesfrench
952 \let\@nineties=\@@ninetiesfrench
953 \@@ordinalstringfrench{#1}{#2}%
954 }
955 \let\@ordinalstringMfrench=\@ordinalstringMfrenchfrance
956 \DeclareRobustCommand{\@ordinalstringFfrenchswiss}[2]{%
957 \let\fc@case\fc@CaseIden
958 \let\fc@first=\fc@@firstFfrench
959 \fc@french@common
960 \let\@seventies=\@@seventiesfrenchswiss
961 \let\@eighties=\@@eightiesfrenchswiss
962 \let\@nineties=\@@ninetiesfrenchswiss
963 \@@ordinalstringfrench{#1}{#2}%
964 }
965 \DeclareRobustCommand{\@ordinalstringFfrenchfrance}[2]{%
966 \let\fc@case\fc@CaseIden
967 \let\fc@first=\fc@@firstFfrench
968 \fc@french@common
969 \let\@seventies=\@@seventiesfrench
970 \let\@eighties=\@@eightiesfrench
971 \let\@nineties=\@@ninetiesfrench
972 \@@ordinalstringfrench{#1}{#2}%
973 }
974 \DeclareRobustCommand{\@ordinalstringFfrenchbelgian}[2]{%
975 \let\fc@case\fc@CaseIden
976 \let\fc@first=\fc@@firstFfrench
977 \fc@french@common
978 \let\@seventies=\@@seventiesfrench
979 \let\@eighties=\@@eightiesfrench
980 \let\@nineties=\@@ninetiesfrench
981 \@@ordinalstringfrench{#1}{#2}%
982 }
983 \let\@ordinalstringFfrench=\@ordinalstringFfrenchfrance
984 \let\@ordinalstringNfrench\@ordinalstringMfrench
985 \DeclareRobustCommand{\@OrdinalstringMfrenchswiss}[2]{%

```

```

986 \let\fc@case\fc@UpperCaseFirstLetter
987 \let\fc@first=\fc@@firstfrench
988 \fc@french@common
989 \let\@seventies=\@@seventiesfrenchswiss
990 \let\@eighties=\@@eightiesfrenchswiss
991 \let\@nineties=\@@ninetiesfrenchswiss
992 \@@ordinalstringfrench{#1}{#2}%
993 }
994 \DeclareRobustCommand{\@OrdinalstringMfrenchfrance}[2]{%
995 \let\fc@case\fc@UpperCaseFirstLetter
996 \let\fc@first=\fc@@firstfrench
997 \fc@french@common
998 \let\@seventies=\@@seventiesfrench
999 \let\@eighties=\@@eightiesfrench
1000 \let\@nineties=\@@ninetiesfrench
1001 \@@ordinalstringfrench{#1}{#2}%
1002 }
1003 \DeclareRobustCommand{\@OrdinalstringMfrenchbelgian}[2]{%
1004 \let\fc@case\fc@UpperCaseFirstLetter
1005 \let\fc@first=\fc@@firstfrench
1006 \fc@french@common
1007 \let\@seventies=\@@seventiesfrench
1008 \let\@eighties=\@@eightiesfrench
1009 \let\@nineties=\@@ninetiesfrench
1010 \@@ordinalstringfrench{#1}{#2}%
1011 }
1012 \let\@OrdinalstringMfrench=\@OrdinalstringMfrenchfrance
1013 \DeclareRobustCommand{\@OrdinalstringFfrenchswiss}[2]{%
1014 \let\fc@case\fc@UpperCaseFirstLetter
1015 \let\fc@first=\fc@@firstfrench
1016 \fc@french@common
1017 \let\@seventies=\@@seventiesfrenchswiss
1018 \let\@eighties=\@@eightiesfrenchswiss
1019 \let\@nineties=\@@ninetiesfrenchswiss
1020 \@@ordinalstringfrench{#1}{#2}%
1021 }
1022 \DeclareRobustCommand{\@OrdinalstringFfrenchfrance}[2]{%
1023 \let\fc@case\fc@UpperCaseFirstLetter
1024 \let\fc@first=\fc@@firstFfrench
1025 \fc@french@common
1026 \let\@seventies=\@@seventiesfrench
1027 \let\@eighties=\@@eightiesfrench
1028 \let\@nineties=\@@ninetiesfrench
1029 \@@ordinalstringfrench{#1}{#2}%
1030 }
1031 \DeclareRobustCommand{\@OrdinalstringFfrenchbelgian}[2]{%
1032 \let\fc@case\fc@UpperCaseFirstLetter
1033 \let\fc@first=\fc@@firstFfrench
1034 \fc@french@common

```



```

1035 \let\@seventies=\@seventiesfrench
1036 \let\@eighties=\@eightiesfrench
1037 \let\@nineties=\@ninetiesfrench
1038 \@ordinalstringfrench{#1}{#2}%
1039 }
1040 \let\@OrdinalstringFfrench=\@OrdinalstringFfrenchfrance
1041 \let\@OrdinalstringNfrench\@OrdinalstringMfrench

```

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

```

1042 \ifcsundef{fc@@do@plural@mark}{}%
1043 {\PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1044   '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.

```

1045 \def\fc@@do@plural@mark#1{%
1046   \ifcase\count10 %
1047     #1% 0=always
1048     \or% 1=never
1049     \or% 2=multiple
1050     \ifnum\count1>1 %
1051       #1%
1052       \fi
1053     \or% 3= multiple g-last
1054     \ifnum\count1>1 %
1055       \ifnum\count0=\count6 %
1056         #1%
1057         \fi
1058       \fi
1059     \or% 4= multiple l-last
1060     \ifnum\count1>1 %
1061       \ifnum\count9=1 %
1062         \else
1063         #1%

```

```

1064     \fi
1065     \fi
1066 \or% 5= multiple lng-last
1067     \ifnum\count1>1 %
1068         \ifnum\count9=1 %
1069             \else
1070                 \if\count0>\count6 %
1071                     #1%
1072             \fi
1073         \fi
1074     \fi
1075 \or% 6= multiple ng-last
1076     \ifnum\count1>1 %
1077         \ifnum\count0>\count6 %
1078             #1%
1079         \fi
1080     \fi
1081 \fi
1082 }

```

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

```

1083 \ifcsundef{fc@@nbrstr@Fpreamble}{}{%
1084     \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1085         'fc@@nbrstr@Fpreamble'}}

```

fc@@nbrstr@Fpreamble

```

1086 \def\fc@@nbrstr@Fpreamble{%
1087     \fc@read@unit{\count1}{0}%
1088     \ifnum\count1=1 %
1089         \let\fc@case@save\fc@case
1090         \def\fc@case{\noexpand\fc@case}%
1091         \def\@nil{\noexpand\@nil}%
1092         \let\fc@nbrstr@postamble\fc@@nbrstr@Fpostamble
1093     \fi
1094 }

```

fc@@nbrstr@Fpostamble

```

1095 \def\fc@@nbrstr@Fpostamble{%
1096     \let\fc@case\fc@case@save
1097     \expandafter\fc@get@last@word\expandafter{\@tempa}\@tempb\@tempc
1098     \def\@tempd{un}%
1099     \ifx\@tempc\@tempd
1100         \let\@tempc\@tempa
1101         \edef\@tempa{\@tempb\fc@case une\@nil}%
1102     \fi
1103 }

```

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

```

1104 \ifcsundef{fc@@pot@longscalefrench}{}{%
1105   \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1106     '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

```

1107 \def\fc@@pot@longscalefrench#1#2#3{%
1108   {%

```

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

```

1109   \edef\@tempb{\number#1}%

```

Let `\count1` be the plural value.

```

1110   \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 .

```

1111   \count2\count0 %
1112   \divide\count2 by 6 %
1113   \count3\count2 %
1114   \multiply\count3 by 6 %
1115   \count3-\count3 %
1116   \advance\count3 by \count0 %
1117   \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)”.

```

1118   \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.

```

1119   \edef\@temp{%
1120     \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.

```

1121     1%
1122     \else
1123     \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`.

```
1124         \ifnum\fc@longscale@nilliard@upto=0 %
```

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

```
1125             2%
```

```
1126         \else
```

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

```
1127             \ifnum\count2>\fc@longscale@nilliard@upto
```

```
1128                 1%
```

```
1129             \else
```

```
1130                 2%
```

```
1131             \fi
```

```
1132         \fi
```

```
1133     \else
```

```
1134         2%
```

```
1135     \fi
```

```
1136 \fi
```

```
1137 }%
```

```
1138 \ifnum\@temph=1 %
```

Here 10^w is formatted as “mil(le)”.

```
1139     \count10=\fc@frenchoptions@mil@plural\space
```

```
1140     \edef\@tempe{%
```

```
1141         \noexpand\fc@case
```

```
1142         mil%
```

```
1143         \fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark
```

```
1144         \noexpand\@nil
```

```
1145     }%
```

```
1146 \else
```

```
1147     % weight >= 6
```

```
1148     \expandafter\fc@@latin@cardinal@pefix\expandafter{\the\count2}\@tempg
```

```
1149     % now form the xxx-illion(s) or xxx-illiard(s) word
```

```
1150     \ifnum\count3>2 %
```

```
1151         \toks10{illiard}%
```

```
1152         \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space
```

```
1153     \else
```

```
1154         \toks10{illion}%
```

```
1155         \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
```

```
1156     \fi
```

```
1157     \edef\@tempe{%
```

```
1158         \noexpand\fc@case
```

```
1159         \@tempg
```

```
1160         \the\toks10 %
```

```
1161         \fc@@do@plural@mark s%
```

```
1162         \noexpand\@nil
```

```
1163     }%
```

```

1164     \fi
1165     \else

```

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

```

1166         \let\@tempe\@empty
1167         \def\@temph{0}%
1168     \fi
1169     \else

```

Case of $w = 0$.

```

1170         \let\@tempe\@empty
1171         \def\@temph{0}%
1172     \fi

```

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

```

1173     \expandafter\toks\expandafter1\expandafter{\@tempe}%
1174     \toks0{#2}%
1175     \edef\@tempa{\the\toks0 \@temph \def\noexpand#3{\the\toks1}}%
1176     \expandafter
1177 } \@tempa
1178 }

```

`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.

```

1179 \ifcsundef{fc@pot@shortscalefrench}{}%
1180 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1181 '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

```

1182 \def\fc@pot@shortscalefrench#1#2#3{%
1183   {%

```

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

```

1184     \edef\@tempb{\number#1}%

```

And let `\count1` be the plural value.

```

1185     \count1=\@tempb

```

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

```
1186 \count2\count0 %
1187 \divide\count2 by 3 %
1188 \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)”.

```
1189 \ifnum\count0>0 % If weight>=3, i.e we do have to append thousand or n-illion(s)/n-illi.
1190 \ifnum\count1>0 % we have at least one thousand/n-illion/n-illiard
1191 \ifnum\count2=0 %
1192 \def\@temph{1}%
1193 \count1=\fc@frenchoptions@mil@plural\space
1194 \edef\@tempe{%
1195 mil%
1196 \fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark
1197 }%
1198 \else
1199 \def\@temph{2}%
1200 % weight >= 6
1201 \expandafter\fc@latin@cardinal@pefix\expandafter{\the\count2}\@tempg
1202 \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
1203 \edef\@tempe{%
1204 \noexpand\fc@case
1205 \@tempg
1206 illion%
1207 \fc@@do@plural@mark s%
1208 \noexpand\@nil
1209 }%
1210 \fi
1211 \else
```

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

```
1212 \def\@temph{0}%
1213 \let\@tempe\@empty
1214 \fi
1215 \else
```

Here $w = 0$.

```
1216 \def\@temph{0}%
1217 \let\@tempe\@empty
1218 \fi
1219 % now place into \@cs{@tempa} the assignment of results \@cs{@temph} and \@cs{@tempe} to to \
1220 % \texttt{\#3} for further propagation after closing brace.
1221 % \begin{macrocode}
1222 \expandafter\toks\expandafter1\expandafter{\@tempe}%
1223 \toks0{\#2}%
1224 \edef\@tempa{\the\toks0 \@temph \def\noexpand#3{\the\toks1}}%
```

```

1225   \expandafter
1226 } \@tempa
1227 }

```

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.

```

1228 \ifcsundef{fc@@pot@recursivefrench}{-}{%
1229   \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1230     '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

```

1231 \def\fc@@pot@recursivefrench#1#2#3{%
1232   {%

```

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

```

1233   \edef\@tempb{\number#1}%
1234   \let\@tempa\@tempa

```

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

```

1235   \count1=\@tempb\space

```

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

```

1236   \ifnum\count1>0 %
1237     \count2\count0 %
1238     \divide\count2 by 9 %
1239     \advance\count2 by -1 %
1240     \let\@tempe\@empty
1241     \edef\@tempf{\fc@frenchoptions@supermillion@dos
1242       de\fc@frenchoptions@supermillion@dos\fc@case milliards\@nil}%
1243     \count11\count0 %
1244     \ifnum\count2>0 %
1245       \count3\count2 %
1246       \count3-\count3 %
1247       \multiply\count3 by 9 %
1248       \advance\count11 by \count3 %
1249     \loop
1250       % (\count2, \count3) <- (\count2 div 2, \count2 mod 2)
1251       \count3\count2 %

```

```

1252         \divide\count3 by 2 %
1253         \multiply\count3 by 2 %
1254         \count3-\count3 %
1255         \advance\count3 by \count2 %
1256         \divide\count2 by 2 %
1257         \ifnum\count3=1 %
1258             \let\@tempg\@tempe
1259             \edef\@tempe{\@tempg\@tempf}%
1260         \fi
1261         \let\@tempg\@tempf
1262         \edef\@tempf{\@tempg\@tempg}%
1263         \ifnum\count2>0 %
1264     \repeat
1265 \fi
1266 \divide\count11 by 3 %
1267 \ifcase\count11 % 0 .. 5
1268 % 0 => d milliard(s) (de milliards)*
1269 \def\@temph{2}%
1270 \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space
1271 \or % 1 => d mille milliard(s) (de milliards)*
1272 \def\@temph{1}%
1273 \count10=\fc@frenchoptions@mil@plural\space
1274 \or % 2 => d million(s) (de milliards)*
1275 \def\@temph{2}%
1276 \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
1277 \or % 3 => d milliard(s) (de milliards)*
1278 \def\@temph{2}%
1279 \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space
1280 \or % 4 => d mille milliards (de milliards)*
1281 \def\@temph{1}%
1282 \count10=\fc@frenchoptions@mil@plural\space
1283 \else % 5 => d million(s) (de milliards)*
1284 \def\@temph{2}%
1285 \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
1286 \fi
1287 \let\@tempg\@tempe
1288 \edef\@tempf{%
1289 \ifcase\count11 % 0 .. 5
1290 \or
1291 mil\fc@@do@plural@mark \fc@frenchoptions@mil@plural@mark
1292 \or
1293 million\fc@@do@plural@mark s%
1294 \or
1295 milliard\fc@@do@plural@mark s%
1296 \or
1297 mil\fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark
1298 \noexpand\@nil\fc@frenchoptions@supermillion@dos
1299 \noexpand\fc@case milliards% 4
1300 \or

```



```

1301         million\fc@@do@plural@mark s%
1302         \noexpand@nil\fc@frenchoptions@supermillion@dos
1303         de\fc@frenchoptions@supermillion@dos\noexpand\fc@case  milliards% 5
1304     \fi
1305 }%
1306 \edef\@tempe{%
1307     \ifx\@tempf\@empty\else
1308     \expandafter\fc@case\@tempf\@nil
1309     \fi
1310     \@tempg
1311 }%
1312 \else
1313     \def\@temph{0}%
1314     \let\@tempe\@empty
1315 \fi

```

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

```

1316     \expandafter\toks\expandafter1\expandafter{\@tempe}%
1317     \toks0{#2}%
1318     \edef\@tempa{\the\toks0 \@temph \def\noexpand#3{\the\toks1}}%
1319     \expandafter
1320 } \@tempa
1321 }

```

`\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.

```

1322 \ifcsundef{fc@muladdfrench}{-}{%
1323     \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1324     '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

```

1325 \def\fc@muladdfrench#1#2#3{%
1326   {%

```

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

```

1327   \edef\@tempc{#1}%
1328   \edef\@tempd{#2}%
1329   \edef\@tempf{#3}%
1330   \let\@tempc\@tempc
1331   \let\@tempd\@tempd

```

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

```

1332   \ifnum\@tempc=1 %
1333     \ifnum\count9=1 %
1334       \let\@tempd\@empty
1335     \fi
1336   \fi

```

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

```

1337   \edef\@tempg{%
1338     \@tempd
1339     \ifx\@tempd\@empty\else
1340       \ifx\@tempf\@empty\else
1341         \ifcase\count9 %
1342         \or
1343         \fc@frenchoptions@submillion@dos
1344         \or
1345         \fc@frenchoptions@supermillion@dos
1346       \fi
1347     \fi
1348   \fi
1349   \@tempf
1350 }%

```

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

```

1351   \edef\@tempf{%
1352     \@tempa
1353     \ifx\@tempa\@empty\else
1354       \ifx\@tempg\@empty\else
1355         \ifcase\count8 %
1356         \or
1357         \fc@frenchoptions@submillion@dos
1358         \or
1359         \fc@frenchoptions@supermillion@dos
1360       \fi

```

```

1361     \fi
1362     \fi
1363     \@tempg
1364 }%
```

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

```

1365     \def\@tempb##1{\def\@tempa{\def\@tempa{##1}}}%
1366     \expandafter\@tempb\expandafter{\@tempb}%
1367     \expandafter
1368 } \@tempa
1369 }%
```

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

```

1370 \ifcsundef{fc@lthundredstringfrench}{}%
1371 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1372   'fc@lthundredstringfrench'}}
```

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

#1 input/output macro to which append the result

Implicit input arguments as follows:

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

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

```

1373 \def\fc@lthundredstringfrench#1{%
1374   {%
```

First save arguments into local temporary macro.

```

1375     \let\@tempc#1%
```

Read units d_w to `\count1`.

```

1376     \fc@read@unit{\count1}{\count0}%
```

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

```

1377     \count3\count0 %
1378     \advance\count3 1 %
1379     \fc@read@unit{\count2}{\count3}%
```

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

```

1380     \edef\@tempa{%
1381       \@tempc
1382       \ifnum\count2>1 %
1383         % 20 .. 99
1384       \ifnum\count2>6 %
1385         % 70 .. 99
1386       \ifnum\count2<8 %
1387         % 70 .. 79
```

```

1388         \@seventies{\count1}%
1389     \else
1390         % 80..99
1391         \ifnum\count2<9 %
1392             % 80 .. 89
1393             \@eighties{\count1}%
1394         \else
1395             % 90 .. 99
1396             \@nineties{\count1}%
1397         \fi
1398     \fi
1399 \else
1400     % 20..69
1401     \@tenstring{\count2}%
1402     \ifnum\count1>0 %
1403         % x1 .. x0
1404         \ifnum\count1=1 %
1405             % x1
1406             \fc@frenchoptions@submillion@dos\@andname\fc@frenchoptions@submillion@dos
1407         \else
1408             % x2 .. x9
1409             -%
1410         \fi
1411         \@unitstring{\count1}%
1412     \fi
1413 \fi
1414 \else
1415     % 0 .. 19
1416     \ifnum\count2=0 % when tens = 0
1417         % 0 .. 9
1418         \ifnum\count1=0 % when units = 0
1419             % \count3=1 when #1 = 0, i.e. only for the unit of the top level number
1420             \ifnum\count3=1 %
1421                 \ifnum\fc@max@weight=0 %
1422                     \@unitstring{0}%
1423                 \fi
1424             \fi
1425         \else
1426             % 1 .. 9
1427             \@unitstring{\count1}%
1428         \fi
1429     \else
1430         % 10 .. 19
1431         \@teenstring{\count1}%
1432     \fi
1433 \fi
1434 }%

```

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

```

1435     \def\@tempb##1{\def\@tempa{\def#1{##1}}}%

```

```

1436   \expandafter\@tempb\expandafter{\@tempa}%
1437   \expandafter
1438 } \@tempa
1439 }

```

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

```

1440 \ifcsundef{fc@ltthousandstringfrench}{}%
1441   \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1442     'fc@ltthousandstringfrench'}%

```

Output is empty for 0. Arguments as follows:

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

Implicit input arguments as follows:

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

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

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

```

1443 \def\fc@ltthousandstringfrench#1{%
1444   {%

```

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

```

1445   \count4\count0 %
1446   \advance\count4 by 2 %
1447   \fc@read@unit{\count2 }{\count4 }%

```

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

```

1448   \advance\count4 by -1 %
1449   \count3\count4 %
1450   \advance\count3 by -1 %
1451   \fc@check@nonzeros{\count3 }{\count4 } \@tempa

```

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

```

1452   \edef\@temps{%
1453     \ifcase\fc@frenchoptions@cent@plural\space
1454     % 0 => always
1455     s%
1456     \or
1457     % 1 => never
1458     \or
1459     % 2 => multiple
1460     \ifnum\count2>1s\fi
1461     \or
1462     % 3 => multiple g-last
1463     \ifnum\count2>1 \ifnum\@tempa=0 \ifnum\count0=\count6s\fi\fi\fi
1464     \or
1465     % 4 => multiple l-last
1466     \ifnum\count2>1 \ifnum\@tempa=0 \ifnum\count9=0s\else\ifnum\count9=2s\fi\fi\fi\fi
1467     \fi

```

```

1468 }%
1469 % compute spacing after cent(s?) into \@tempb
1470 \expandafter\let\expandafter\@tempb
1471 \ifnum\@tempa>0 \fc@frenchoptions@submillion@dos\else\@empty\fi
1472 % now place into \@tempa the hundreds
1473 \edef\@tempa{%
1474 \ifnum\count2=0 %
1475 \else
1476 \ifnum\count2=1 %
1477 \expandafter\fc@case\@hundred\@nil
1478 \else
1479 \@unitstring{\count2}\fc@frenchoptions@submillion@dos
1480 \noexpand\fc@case\@hundred\@temps\noexpand\@nil
1481 \fi
1482 \@tempb
1483 \fi
1484 }%
1485 % now append to \@tempa the ten and unit
1486 \fc@lthundredstringfrench\@tempa

```

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

```

1487 \def\@tempb##1{\def\@tempa{\def#1{##1}}}%
1488 \expandafter\@tempb\expandafter{\@tempa}%
1489 \expandafter
1490 }\@tempa
1491 }

```

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

```

1492 \ifcsundef{\@numberstringfrench}{\}%
1493 \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

```

1494 \def\@numberstringfrench#1#2{%
1495 {%

```

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

```

1496 \edef\@tempa{#1}%
1497 \expandafter\fc@number@parser\expandafter{\@tempa}%
1498 \ifnum\fc@min@weight<0 %
1499 \PackageError{fmtcount}{Out of range}%
1500 {This macro does not work with fractional numbers}%
1501 \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'.

```

1502 \edef\@tempa{\ifcase\fc@sign@case\space\or\fc@case plus\@nil\or\fc@case moins\@nil\fi}%

```

```

1503 \fc@nbrstr@preamble
1504 \fc@@nbrstrfrench@inner
1505 \fc@nbrstr@postamble

```

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

```

1506 \def\@tempb##1{\def\@tempa{\def#2{##1}}}%
1507 \expandafter\@tempb\expandafter{\@tempa}%
1508 \expandafter
1509 }\@tempa
1510 }

```

`\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.

```

1511 \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`.

```

1512 \count0=\fc@max@weight
1513 \divide\count0 by 3 %
1514 \multiply\count0 by 3 %

```

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

```

1515 \fc@intpart@find@last{\count5 }%
1516 \count6\count5 %
1517 \divide\count6 3 %
1518 \multiply\count6 3 %
1519 \count8=0 %
1520 \loop

```

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

```

1521 \count1\count0 %
1522 \advance\count1 by 2 %
1523 \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`.

```

1524 \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`

```

1525 \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 .

```
1526      \fc@muladdfrench\@tempt\@tempd\@tempb
```

Then iterate.

```
1527      \count8\count9 %
1528      \advance\count0 by -3 %
1529      \ifnum\count6>\count0 \else
1530      \repeat
1531 }
```

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

```
1532 \ifcsundef{@@ordinalstringfrench}{-}{%
1533   \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1534   '@@ordinalstringfrench'}}
```

Arguments are as follows:

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

```
1535 \def\@@ordinalstringfrench#1#2{%
1536   {%
```

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

```
1537   \edef\@tempa{#1}%
1538   \expandafter\fc@number@parser\expandafter{\@tempa}%
1539   \ifnum\fc@min@weight<0 %
1540     \PackageError{fmtcount}{Out of range}%
1541     {This macro does not work with fractional numbers}%
1542   \fi
1543   \ifnum\fc@sign@case>0 %
1544     \PackageError{fmtcount}{Out of range}%
1545     {This macro does with negative or explicitly marked as positive numbers}%
1546   \fi
```

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

```
1547   \ifnum\fc@max@weight=0 %
1548     \ifnum\csname fc@digit@0\endcsname=1 %
1549       \count0=1 %
1550     \else
1551       \count0=0 %
1552     \fi
1553   \else
1554     \count0=0 %
1555   \fi
1556   \ifnum\count0=1 %

1557     \protected@edef\@tempa{\expandafter\fc@case\fc@first\@nil}%
1558   \else
```


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

```

1559     \def\@tempa##1{%
1560         \expandafter\edef\csname fc@frenchoptions@##1@plural\endcsname{%
1561             \ifcase\csname fc@frenchoptions@##1@plural\endcsname\space
1562                 0% 0: always => always
1563                 \or
1564                 1% 1: never => never
1565                 \or
1566                 6% 2: multiple => multiple ng-last
1567                 \or
1568                 1% 3: multiple g-last => never
1569                 \or
1570                 5% 4: multiple l-last => multiple lng-last
1571                 \or
1572                 5% 5: multiple lng-last => multiple lng-last
1573                 \or
1574                 6% 6: multiple ng-last => multiple ng-last
1575             \fi
1576         }%
1577     }%
1578     \@tempa{vingt}%
1579     \@tempa{cent}%
1580     \@tempa{mil}%
1581     \@tempa{n-illion}%
1582     \@tempa{n-illiard}%

```

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

```

1583     \let\fc@case@save\fc@case
1584     \def\fc@case{\noexpand\fc@case}%
1585     \def\@nil{\noexpand\@nil}%

```

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

```

1586     \let\@tempa\@empty
1587     \fc@nbrstrfrench@inner

```

Now restore \fc@case

```

1588     \let\fc@case\fc@case@save

```

Now we add the “ième” ending

```

1589     \expandafter\fc@get@last@word\expandafter{\@tempa}\@tempb\@tempc
1590     \expandafter\fc@get@last@letter\expandafter{\@tempc}\@tempd\@tempe
1591     \def\@tempf{e}%
1592     \ifx\@tempe\@tempf
1593         \protected@edef\@tempa{\@tempb\expandafter\fc@case\@tempd i\protect\‘eme\@nil}%
1594     \else
1595         \def\@tempf{q}%
1596         \ifx\@tempe\@tempf
1597             \protected@edef\@tempa{\@tempb\expandafter\fc@case\@tempd qui\protect\‘eme\@nil}%
1598         \else
1599             \def\@tempf{f}%

```

```

1600         \ifx\@tempe\@tempf
1601         \protected@edef\@tempa{\@tempb\expandafter\fc@case\@tempd vi\protect\‘eme\@nil}
1602         \else
1603         \protected@edef\@tempa{\@tempb\expandafter\fc@case\@tempc i\protect\‘eme\@nil}%
1604         \fi
1605     \fi
1606 \fi
1607 \fi

```

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

```

1608     \def\@tempb##1{\def\@tempa{\def#2{##1}}}%
1609     \expandafter\@tempb\expandafter{\@tempa}%
1610     \expandafter
1611 } \@tempa
1612 }

```

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

```

1613 \newcommand*\fc@frenchoptions@setdefaults{%
1614   \csname KV@fcfrench@all plural\endcsname{reformed}%
1615   \def\fc@frenchoptions@submillion@dos{-}%
1616   \let\fc@frenchoptions@supermillion@dos\space
1617   \let\fc@u@in@duo\@empty% Could be ‘u’
1618   % \let\fc@poweroften\fc@pot@longscalefrench
1619   \let\fc@poweroften\fc@pot@recursivefrench
1620   \def\fc@longscale@illiard@upto{0}% infinity
1621   \def\fc@frenchoptions@mil@plural@mark{le}%
1622 }
1623 \fc@frenchoptions@setdefaults

```

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

```

1624 \def\@ordinalstringMfrench{\csuse{@ordinalstringMfrench\fmtcount@french}}%
1625 \def\@ordinalstringFfrench{\csuse{@ordinalstringFfrench\fmtcount@french}}%
1626 \def\@OrdinalstringMfrench{\csuse{@OrdinalstringMfrench\fmtcount@french}}%
1627 \def\@OrdinalstringFfrench{\csuse{@OrdinalstringFfrench\fmtcount@french}}%
1628 \def\@numberstringMfrench{\csuse{@numberstringMfrench\fmtcount@french}}%
1629 \def\@numberstringFfrench{\csuse{@numberstringFfrench\fmtcount@french}}%
1630 \def\@NumberstringMfrench{\csuse{@NumberstringMfrench\fmtcount@french}}%
1631 \def\@NumberstringFfrench{\csuse{@NumberstringFfrench\fmtcount@french}}%

```

9.0.6 fc-frenchb.def

```

1632 \ProvidesFCLanguage{frenchb}[2013/08/17]%
1633 \FCloadlang{french}%

```

Set frenchb to be equivalent to french.

```

1634 \global\let\@ordinalMfrenchb=\@ordinalMfrench
1635 \global\let\@ordinalFfrenchb=\@ordinalFfrench
1636 \global\let\@ordinalNfrenchb=\@ordinalNfrench

```

```

1637 \global\let\@numberstringMfrenchb=\@numberstringMfrench
1638 \global\let\@numberstringFfrenchb=\@numberstringFfrench
1639 \global\let\@numberstringNfrenchb=\@numberstringNfrench
1640 \global\let\@NumberstringMfrenchb=\@NumberstringMfrench
1641 \global\let\@NumberstringFfrenchb=\@NumberstringFfrench
1642 \global\let\@NumberstringNfrenchb=\@NumberstringNfrench
1643 \global\let\@ordinalstringMfrenchb=\@ordinalstringMfrench
1644 \global\let\@ordinalstringFfrenchb=\@ordinalstringFfrench
1645 \global\let\@ordinalstringNfrenchb=\@ordinalstringNfrench
1646 \global\let\@OrdinalstringMfrenchb=\@OrdinalstringMfrench
1647 \global\let\@OrdinalstringFfrenchb=\@OrdinalstringFfrench
1648 \global\let\@OrdinalstringNfrenchb=\@OrdinalstringNfrench

```

9.0.7 fc-german.def

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

```
1649 \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:

```

1650 \newcommand{\@ordinalMgerman}[2]{%
1651   \edef#2{\number#1\relax.}%
1652 }%
1653 \global\let\@ordinalMgerman\@ordinalMgerman

```

Feminine:

```

1654 \newcommand{\@ordinalFgerman}[2]{%
1655   \edef#2{\number#1\relax.}%
1656 }%
1657 \global\let\@ordinalFgerman\@ordinalFgerman

```

Neuter:

```

1658 \newcommand{\@ordinalNgerman}[2]{%
1659   \edef#2{\number#1\relax.}%
1660 }%
1661 \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):

```

1662 \newcommand*\@unitstringgerman[1]{%
1663   \ifcase#1%
1664     null%
1665     \or ein%
1666     \or zwei%
1667     \or drei%
1668     \or vier%
1669     \or f\"unf%
1670     \or sechs%
1671     \or sieben%

```

```

1672 \or acht%
1673 \or neun%
1674 \fi
1675 }%
1676 \global\let\@@unitstringgerman\@@unitstringgerman

```

Tens (argument must go from 1 to 10):

```

1677 \newcommand*\@@tenstringgerman[1]{%
1678 \ifcase#1%
1679 \or zehn%
1680 \or zwanzig%
1681 \or drei{\ss}ig%
1682 \or vierzig%
1683 \or f\unfzig%
1684 \or sechzig%
1685 \or siebzig%
1686 \or achtzig%
1687 \or neunzig%
1688 \or einhundert%
1689 \fi
1690 }%
1691 \global\let\@@tenstringgerman\@@tenstringgerman

```

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

```

1692 \providecommand*\einhundert{einhundert}%
1693 \providecommand*\eintausend{eintausend}%
1694 \global\let\einhundert\einhundert
1695 \global\let\eintausend\eintausend

```

Teens:

```

1696 \newcommand*\@@teenstringgerman[1]{%
1697 \ifcase#1%
1698 zehn%
1699 \or elf%
1700 \or zw\olf%
1701 \or dreizehn%
1702 \or vierzehn%
1703 \or f\unfzehn%
1704 \or sechzehn%
1705 \or siebzehn%
1706 \or achtzehn%
1707 \or neunzehn%
1708 \fi
1709 }%
1710 \global\let\@@teenstringgerman\@@teenstringgerman

```

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

```

1711 \DeclareRobustCommand{\@numberstringMgerman}[2]{%
1712 \let\@unitstring=\@@unitstringgerman
1713 \let\@teenstring=\@@teenstringgerman

```

```

1714 \let\@tenstring=\@tenstringgerman
1715 \@numberstringgerman{#1}{#2}%
1716 }%
1717 \global\let\@numberstringMgerman\@numberstringMgerman

  Feminine and neuter forms:
1718 \global\let\@numberstringFgerman=\@numberstringMgerman
1719 \global\let\@numberstringNgerman=\@numberstringMgerman

  As above, but initial letters in upper case:
1720 \DeclareRobustCommand{\@NumberstringMgerman}[2]{%
1721 \@numberstringMgerman{#1}{\@num@str}%
1722 \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
1723 }%
1724 \global\let\@NumberstringMgerman\@NumberstringMgerman

  Feminine and neuter form:
1725 \global\let\@NumberstringFgerman=\@NumberstringMgerman
1726 \global\let\@NumberstringNgerman=\@NumberstringMgerman

  As above, but for ordinals.
1727 \DeclareRobustCommand{\@OrdinalstringMgerman}[2]{%
1728 \let\@unitthstring=\@unitthstringMgerman
1729 \let\@teenthstring=\@teenthstringMgerman
1730 \let\@tenthstring=\@tenthstringMgerman
1731 \let\@unitstring=\@unitstringgerman
1732 \let\@teenstring=\@teenstringgerman
1733 \let\@tenstring=\@tenstringgerman
1734 \def\@thousandth{tausendster}%
1735 \def\@hundredth{hundertster}%
1736 \@ordinalstringgerman{#1}{#2}%
1737 }%
1738 \global\let\@ordinalstringMgerman\@ordinalstringMgerman

  Feminine form:
1739 \DeclareRobustCommand{\@OrdinalstringFgerman}[2]{%
1740 \let\@unitthstring=\@unitthstringFgerman
1741 \let\@teenthstring=\@teenthstringFgerman
1742 \let\@tenthstring=\@tenthstringFgerman
1743 \let\@unitstring=\@unitstringgerman
1744 \let\@teenstring=\@teenstringgerman
1745 \let\@tenstring=\@tenstringgerman
1746 \def\@thousandth{tausendste}%
1747 \def\@hundredth{hundertste}%
1748 \@ordinalstringgerman{#1}{#2}%
1749 }%
1750 \global\let\@ordinalstringFgerman\@ordinalstringFgerman

  Neuter form:
1751 \DeclareRobustCommand{\@OrdinalstringNgerman}[2]{%
1752 \let\@unitthstring=\@unitthstringNgerman
1753 \let\@teenthstring=\@teenthstringNgerman

```

```

1754 \let\@tenthstring=\@tenthstringNgerman
1755 \let\@unitstring=\@unitstringgerman
1756 \let\@teenstring=\@teenstringgerman
1757 \let\@tenstring=\@tenstringgerman
1758 \def\@thousandth{tausendstes}%
1759 \def\@hundredth{hunderstes}%
1760 \@ordinalstringgerman{#1}{#2}%
1761 }%
1762 \global\let\@ordinalstringNgerman\@ordinalstringNgerman

```

As above, but with initial letters in upper case.

```

1763 \DeclareRobustCommand{\@OrdinalstringMgerman}[2]{%
1764 \@ordinalstringMgerman{#1}{\@num@str}%
1765 \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
1766 }%
1767 \global\let\@OrdinalstringMgerman\@OrdinalstringMgerman

```

Feminine form:

```

1768 \DeclareRobustCommand{\@OrdinalstringFgerman}[2]{%
1769 \@ordinalstringFgerman{#1}{\@num@str}%
1770 \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
1771 }%
1772 \global\let\@OrdinalstringFgerman\@OrdinalstringFgerman

```

Neuter form:

```

1773 \DeclareRobustCommand{\@OrdinalstringNgerman}[2]{%
1774 \@ordinalstringNgerman{#1}{\@num@str}%
1775 \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
1776 }%
1777 \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:

```

1778 \newcommand*\@unitthstringMgerman[1]{%
1779 \ifcase#1%
1780 nullter%
1781 \or erster%
1782 \or zweiter%
1783 \or dritter%
1784 \or vierter%
1785 \or f"unfter%
1786 \or sechster%
1787 \or siebter%
1788 \or achter%
1789 \or neunter%
1790 \fi
1791 }%
1792 \global\let\@unitthstringMgerman\@unitthstringMgerman

```

Tens:

```

1793 \newcommand*\@tenthstringMgerman[1]{%

```

```

1794 \ifcase#1%
1795 \or zehnter%
1796 \or zwanzigster%
1797 \or drei{\ss}igster%
1798 \or vierzigster%
1799 \or f\"unfzigster%
1800 \or sechzigster%
1801 \or siebzigster%
1802 \or achtzigster%
1803 \or neunzigster%
1804 \fi
1805 }%
1806 \global\let\@@tenthstringMgerman\@@tenthstringMgerman

  Teens:
1807 \newcommand*\@@teenthstringMgerman[1]{%
1808 \ifcase#1%
1809 zehnter%
1810 \or elfter%
1811 \or zw\"olfter%
1812 \or dreizehnter%
1813 \or vierzehnter%
1814 \or f\"unfzehnter%
1815 \or sechzehnter%
1816 \or siebzehnter%
1817 \or achtzehnter%
1818 \or neunzehnter%
1819 \fi
1820 }%
1821 \global\let\@@teenthstringMgerman\@@teenthstringMgerman

  Units (feminine):
1822 \newcommand*\@@unitthstringFgerman[1]{%
1823 \ifcase#1%
1824 nullte%
1825 \or erste%
1826 \or zweite%
1827 \or dritte%
1828 \or vierte%
1829 \or f\"unfte%
1830 \or sechste%
1831 \or siebte%
1832 \or achte%
1833 \or neunte%
1834 \fi
1835 }%
1836 \global\let\@@unitthstringFgerman\@@unitthstringFgerman

  Tens (feminine):
1837 \newcommand*\@@tenthstringFgerman[1]{%
1838 \ifcase#1%

```

```

1839 \or zehnte%
1840 \or zwanzigste%
1841 \or drei{\ss}igste%
1842 \or vierzigste%
1843 \or f\"unfzigste%
1844 \or sechzigste%
1845 \or siebzigste%
1846 \or achtzigste%
1847 \or neunzigste%
1848 \fi
1849 }%
1850 \global\let\@@tenthstringFgerman\@@tenthstringFgerman
      Teens (feminine)
1851 \newcommand*\@@teenthstringFgerman[1]{%
1852 \ifcase#1%
1853 zehnte%
1854 \or elfte%
1855 \or zw\"olfte%
1856 \or dreizehnte%
1857 \or vierzehnte%
1858 \or f\"unfzehnte%
1859 \or sechzehnte%
1860 \or siebzehnte%
1861 \or achtzehnte%
1862 \or neunzehnte%
1863 \fi
1864 }%
1865 \global\let\@@teenthstringFgerman\@@teenthstringFgerman
      Units (neuter):
1866 \newcommand*\@@unitthstringNgerman[1]{%
1867 \ifcase#1%
1868 nulltes%
1869 \or erstes%
1870 \or zweites%
1871 \or drittes%
1872 \or viertes%
1873 \or f\"unftes%
1874 \or sechstes%
1875 \or siebtes%
1876 \or achttes%
1877 \or neuntes%
1878 \fi
1879 }%
1880 \global\let\@@unitthstringNgerman\@@unitthstringNgerman
      Tens (neuter):
1881 \newcommand*\@@tenthstringNgerman[1]{%
1882 \ifcase#1%
1883 \or zehntes%

```



```

1884 \or zwanzigstes%
1885 \or drei{\ss}igstes%
1886 \or vierzigstes%
1887 \or f\"unfzigstes%
1888 \or sechzigstes%
1889 \or siebzigstes%
1890 \or achtzigstes%
1891 \or neunzigstes%
1892 \fi
1893 }%
1894 \global\let\@@tenthstringNgerman\@@tenthstringNgerman

```

Teens (neuter)

```

1895 \newcommand*\@@teenthstringNgerman[1]{%
1896 \ifcase#1%
1897 zehntes%
1898 \or elftes%
1899 \or zw\"olftes%
1900 \or dreizehntes%
1901 \or vierzehntes%
1902 \or f\"unfzehntes%
1903 \or sechzehntes%
1904 \or siebzehntes%
1905 \or achtzehntes%
1906 \or neunzehntes%
1907 \fi
1908 }%
1909 \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.

```

1910 \newcommand*\@@numberunderhundredgerman[2]{%
1911 \ifnum#1<10\relax
1912 \ifnum#1>0\relax
1913 \eappto#2{\@unitstring{#1}}%
1914 \fi
1915 \else
1916 \@tmpstrctr=#1\relax
1917 \@FCmodulo{\@tmpstrctr}{10}%
1918 \ifnum#1<20\relax
1919 \eappto#2{\@teenstring{\@tmpstrctr}}%
1920 \else
1921 \ifnum\@tmpstrctr=0\relax
1922 \else
1923 \eappto#2{\@unitstring{\@tmpstrctr}und}%
1924 \fi
1925 \@tmpstrctr=#1\relax
1926 \divide\@tmpstrctr by 10\relax
1927 \eappto#2{\@tenstring{\@tmpstrctr}}%
1928 \fi

```

```

1929 \fi
1930 }%
1931 \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.

1932 \newcommand*\@@numberstringgerman[2]{%
1933 \ifnum#1>99999\relax
1934   \PackageError{fmtcount}{Out of range}%
1935   {This macro only works for values less than 100000}%
1936 \else
1937   \ifnum#1<0\relax
1938     \PackageError{fmtcount}{Negative numbers not permitted}%
1939     {This macro does not work for negative numbers, however
1940     you can try typing "minus" first, and then pass the modulus of
1941     this number}%
1942   \fi
1943 \fi
1944 \def#2{}%
1945 \@strctr=#1\relax \divide\@strctr by 1000\relax
1946 \ifnum\@strctr>1\relax

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

1947 \@@numberunderhundredgerman{\@strctr}{#2}%
1948 \appto#2{tausend}%
1949 \else

  #1 lies in range [1000,1999]

1950 \ifnum\@strctr=1\relax
1951   \eappto#2{eintausend}%
1952 \fi
1953 \fi
1954 \@strctr=#1\relax
1955 \@FCmodulo{\@strctr}{1000}%
1956 \divide\@strctr by 100\relax
1957 \ifnum\@strctr>1\relax

  now dealing with number in range [200,999]

1958 \eappto#2{\@unitstring{\@strctr}hundert}%
1959 \else
1960   \ifnum\@strctr=1\relax

    dealing with number in range [100,199]

1961     \ifnum#1>1000\relax

      if original number > 1000, use einhundert

1962       \appto#2{einhundert}%
1963     \else

      otherwise use \einhundert

1964       \eappto#2{einhundert}%
1965     \fi

```

```

1966 \fi
1967 \fi
1968 \@strctr=#1\relax
1969 \@FCmodulo{\@strctr}{100}%
1970 \ifnum#1=0\relax
1971 \def#2{null}%
1972 \else
1973 \ifnum\@strctr=1\relax
1974 \appto#2{eins}%
1975 \else
1976 \@@numberunderhundredgerman{\@strctr}{#2}%
1977 \fi
1978 \fi
1979 }%
1980 \global\let\@@numberstringgerman\@@numberstringgerman

```

As above, but for ordinals

```

1981 \newcommand*\@@numberunderhundredthgerman[2]{%
1982 \ifnum#1<10\relax
1983 \eappto#2{\@unitthstring{#1}}%
1984 \else
1985 \@tmpstrctr=#1\relax
1986 \@FCmodulo{\@tmpstrctr}{10}%
1987 \ifnum#1<20\relax
1988 \eappto#2{\@teenthstring{\@tmpstrctr}}%
1989 \else
1990 \ifnum\@tmpstrctr=0\relax
1991 \else
1992 \eappto#2{\@unitstring{\@tmpstrctr}und}%
1993 \fi
1994 \@tmpstrctr=#1\relax
1995 \divide\@tmpstrctr by 10\relax
1996 \eappto#2{\@tenthstring{\@tmpstrctr}}%
1997 \fi
1998 \fi
1999 }%
2000 \global\let\@@numberunderhundredthgerman\@@numberunderhundredthgerman

2001 \newcommand*\@@ordinalstringgerman[2]{%
2002 \ifnum#1>99999\relax
2003 \PackageError{fmtcount}{Out of range}%
2004 {This macro only works for values less than 100000}%
2005 \else
2006 \ifnum#1<0\relax
2007 \PackageError{fmtcount}{Negative numbers not permitted}%
2008 {This macro does not work for negative numbers, however
2009 you can try typing "minus" first, and then pass the modulus of
2010 this number}%
2011 \fi
2012 \fi

```

```

2013 \def#2{}%
2014 \@strctr=#1\relax \divide\@strctr by 1000\relax
2015 \ifnum\@strctr>1\relax

  #1 is  $\geq 2000$ , \@strctr now contains the number of thousands
2016 \@@numberunderhundredgerman{\@strctr}{#2}%

  is that it, or is there more?
2017 \@tmpstrctr=#1\relax \@FCmodulo{\@tmpstrctr}{1000}%
2018 \ifnum\@tmpstrctr=0\relax
2019 \eappto#2{\@thousandth}%
2020 \else
2021 \appto#2{tausend}%
2022 \fi
2023 \else

  #1 lies in range [1000,1999]
2024 \ifnum\@strctr=1\relax
2025 \ifnum#1=1000\relax
2026 \eappto#2{\@thousandth}%
2027 \else
2028 \eappto#2{\eintausend}%
2029 \fi
2030 \fi
2031 \fi
2032 \@strctr=#1\relax
2033 \@FCmodulo{\@strctr}{1000}%
2034 \divide\@strctr by 100\relax
2035 \ifnum\@strctr>1\relax

  now dealing with number in range [200,999] is that it, or is there more?
2036 \@tmpstrctr=#1\relax \@FCmodulo{\@tmpstrctr}{100}%
2037 \ifnum\@tmpstrctr=0\relax
2038 \ifnum\@strctr=1\relax
2039 \eappto#2{\@hundredth}%
2040 \else
2041 \eappto#2{\@unitstring{\@strctr}\@hundredth}%
2042 \fi
2043 \else
2044 \eappto#2{\@unitstring{\@strctr}hundert}%
2045 \fi
2046 \else
2047 \ifnum\@strctr=1\relax

  dealing with number in range [100,199] is that it, or is there more?
2048 \@tmpstrctr=#1\relax \@FCmodulo{\@tmpstrctr}{100}%
2049 \ifnum\@tmpstrctr=0\relax
2050 \eappto#2{\@hundredth}%
2051 \else
2052 \ifnum#1>1000\relax
2053 \appto#2{einhundert}%

```

```

2054     \else
2055         \eappto#2{\einhundert}%
2056     \fi
2057 \fi
2058 \fi
2059 \fi
2060 \@strctr=#1\relax
2061 \@FCmodulo{\@strctr}{100}%
2062 \ifthenelse{\@strctr=0 \and #1>0}{\}{%
2063 \@@numberunderhundredthgerman{\@strctr}{#2}%
2064 }%
2065 }%
2066 \global\let\@@ordinalstringgerman\@@ordinalstringgerman

    Load fc-germanb.def if not already loaded
2067 \FCloadlang{germanb}%

```

9.0.8 fc-germanb.def

```

2068 \ProvidesFCLanguage{germanb}[2013/08/17]%

```

Load fc-german.def if not already loaded

```

2069 \FCloadlang{german}%

```

Set germanb to be equivalent to german.

```

2070 \global\let\@ordinalMgermanb=\@ordinalMgerman
2071 \global\let\@ordinalFgermanb=\@ordinalFgerman
2072 \global\let\@ordinalNgermanb=\@ordinalNgerman
2073 \global\let\@numberstringMgermanb=\@numberstringMgerman
2074 \global\let\@numberstringFgermanb=\@numberstringFgerman
2075 \global\let\@numberstringNgermanb=\@numberstringNgerman
2076 \global\let\@NumberstringMgermanb=\@NumberstringMgerman
2077 \global\let\@NumberstringFgermanb=\@NumberstringFgerman
2078 \global\let\@NumberstringNgermanb=\@NumberstringNgerman
2079 \global\let\@ordinalstringMgermanb=\@ordinalstringMgerman
2080 \global\let\@ordinalstringFgermanb=\@ordinalstringFgerman
2081 \global\let\@ordinalstringNgermanb=\@ordinalstringNgerman
2082 \global\let\@OrdinalstringMgermanb=\@OrdinalstringMgerman
2083 \global\let\@OrdinalstringFgermanb=\@OrdinalstringFgerman
2084 \global\let\@OrdinalstringNgermanb=\@OrdinalstringNgerman

```

9.0.9 fc-italian

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

```

2085 \ProvidesFCLanguage{italian}[2013/08/17]
2086
2087 \RequirePackage{itnumpar}
2088
2089 \newcommand{\@numberstringMitalian}[2]{%
2090     \edef#2{\noexpand\printnumeroinparole{#1}}%

```

```

2091 }
2092 \global\let\@numberstringMitalian\@numberstringMitalian
2093
2094 \newcommand{\@numberstringFitalian}[2]{%
2095   \edef#2{\noexpand\printnumeroinparole{#1}}}
2096
2097 \global\let\@numberstringFitalian\@numberstringFitalian
2098
2099 \newcommand{\@NumberstringMitalian}[2]{%
2100   \edef#2{\noexpand\printNumeroinparole{#1}}%
2101 }
2102 \global\let\@NumberstringMitalian\@NumberstringMitalian
2103
2104 \newcommand{\@NumberstringFitalian}[2]{%
2105   \edef#2{\noexpand\printNumeroinparole{#1}}%
2106 }
2107 \global\let\@NumberstringFitalian\@NumberstringFitalian
2108
2109 \newcommand{\@ordinalstringMitalian}[2]{%
2110   \edef#2{\noexpand\printordinalem{#1}}%
2111 }
2112 \global\let\@ordinalstringMitalian\@ordinalstringMitalian
2113
2114 \newcommand{\@ordinalstringFitalian}[2]{%
2115   \edef#2{\noexpand\printordinalef{#1}}%
2116 }
2117 \global\let\@ordinalstringFitalian\@ordinalstringFitalian
2118
2119 \newcommand{\@OrdinalstringMitalian}[2]{%
2120   \edef#2{\noexpand\printOrdinalem{#1}}%
2121 }
2122 \global\let\@OrdinalstringMitalian\@OrdinalstringMitalian
2123
2124 \newcommand{\@OrdinalstringFitalian}[2]{%
2125   \edef#2{\noexpand\printOrdinalef{#1}}%
2126 }
2127 \global\let\@OrdinalstringFitalian\@OrdinalstringFitalian
2128
2129 \newcommand{\@ordinalMitalian}[2]{%
2130   \edef#2{#1\relax\noexpand\fmtord{o}}}
2131
2132 \global\let\@ordinalMitalian\@ordinalMitalian
2133
2134 \newcommand{\@ordinalFitalian}[2]{%
2135   \edef#2{#1\relax\noexpand\fmtord{a}}}
2136 \global\let\@ordinalFitalian\@ordinalFitalian

```

9.0.10 fc-ngerman.def

```

2137 \ProvidesFCLanguage{ngerman}[2012/06/18]%
2138 \FCloadlang{german}%
2139 \FCloadlang{ngermanb}%

```

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

```

2140 \global\let\@ordinalMngerman=\@ordinalMgerman
2141 \global\let\@ordinalFngerman=\@ordinalFgerman
2142 \global\let\@ordinalNngerman=\@ordinalNgerman
2143 \global\let\@numberstringMngerman=\@numberstringMgerman
2144 \global\let\@numberstringFngerman=\@numberstringFgerman
2145 \global\let\@numberstringNngerman=\@numberstringNgerman
2146 \global\let\@NumberstringMngerman=\@NumberstringMgerman
2147 \global\let\@NumberstringFngerman=\@NumberstringFgerman
2148 \global\let\@NumberstringNngerman=\@NumberstringNgerman
2149 \global\let\@ordinalstringMngerman=\@ordinalstringMgerman
2150 \global\let\@ordinalstringFngerman=\@ordinalstringFgerman
2151 \global\let\@ordinalstringNngerman=\@ordinalstringNgerman
2152 \global\let\@OrdinalstringMngerman=\@OrdinalstringMgerman
2153 \global\let\@OrdinalstringFngerman=\@OrdinalstringFgerman
2154 \global\let\@OrdinalstringNngerman=\@OrdinalstringNgerman

```

9.0.11 fc-ngermanb.def

```

2155 \ProvidesFCLanguage{ngermanb}[2013/08/17]%
2156 \FCloadlang{german}%

```

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

```

2157 \global\let\@ordinalMngermanb=\@ordinalMgerman
2158 \global\let\@ordinalFngermanb=\@ordinalFgerman
2159 \global\let\@ordinalNngermanb=\@ordinalNgerman
2160 \global\let\@numberstringMngermanb=\@numberstringMgerman
2161 \global\let\@numberstringFngermanb=\@numberstringFgerman
2162 \global\let\@numberstringNngermanb=\@numberstringNgerman
2163 \global\let\@NumberstringMngermanb=\@NumberstringMgerman
2164 \global\let\@NumberstringFngermanb=\@NumberstringFgerman
2165 \global\let\@NumberstringNngermanb=\@NumberstringNgerman
2166 \global\let\@ordinalstringMngermanb=\@ordinalstringMgerman
2167 \global\let\@ordinalstringFngermanb=\@ordinalstringFgerman
2168 \global\let\@ordinalstringNngermanb=\@ordinalstringNgerman
2169 \global\let\@OrdinalstringMngermanb=\@OrdinalstringMgerman
2170 \global\let\@OrdinalstringFngermanb=\@OrdinalstringFgerman
2171 \global\let\@OrdinalstringNngermanb=\@OrdinalstringNgerman

```

Load fc-ngerman.def if not already loaded

```

2172 \FCloadlang{ngerman}%

```

9.0.12 fc-portuges.def

Portuguse definitions

```

2173 \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:

```

2174 \newcommand*\@ordinalMportuges[2]{%
2175   \ifnum#1=0\relax
2176     \edef#2{\number#1}%
2177   \else
2178     \edef#2{\number#1\relax\noexpand\fmtord{o}}%
2179   \fi
2180 }%
2181 \global\let\@ordinalMportuges\@ordinalMportuges

```

Feminine:

```

2182 \newcommand*\@ordinalFportuges[2]{%
2183   \ifnum#1=0\relax
2184     \edef#2{\number#1}%
2185   \else
2186     \edef#2{\number#1\relax\noexpand\fmtord{a}}%
2187   \fi
2188 }%
2189 \global\let\@ordinalFportuges\@ordinalFportuges

```

Make neuter same as masculine:

```

2190 \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):

```

2191 \newcommand*\@@unitstringportuges[1]{%
2192   \ifcase#1\relax
2193     zero%
2194     \or um%
2195     \or dois%
2196     \or tr\^es%
2197     \or quatro%
2198     \or cinco%
2199     \or seis%
2200     \or sete%
2201     \or oito%
2202     \or nove%
2203   \fi
2204 }%
2205 \global\let\@@unitstringportuges\@@unitstringportuges
2206 % \end{macrocode}
2207 % As above, but for feminine:
2208 % \begin{macrocode}
2209 \newcommand*\@@unitstringFportuges[1]{%
2210   \ifcase#1\relax
2211     zero%
2212     \or uma%

```



```

2213 \or duas%
2214 \or tr\^es%
2215 \or quatro%
2216 \or cinco%
2217 \or seis%
2218 \or sete%
2219 \or oito%
2220 \or nove%
2221 \fi
2222 }%
2223 \global\let\@@unitstringFportuges\@@unitstringFportuges

```

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

```

2224 \newcommand*\@@tenstringportuges[1]{%
2225 \ifcase#1\relax
2226 \or dez%
2227 \or vinte%
2228 \or trinta%
2229 \or quarenta%
2230 \or cinq\ "uenta%
2231 \or sessenta%
2232 \or setenta%
2233 \or oitenta%
2234 \or noventa%
2235 \or cem%
2236 \fi
2237 }%
2238 \global\let\@@tenstringportuges\@@tenstringportuges

```

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

```

2239 \newcommand*\@@teenstringportuges[1]{%
2240 \ifcase#1\relax
2241 dez%
2242 \or onze%
2243 \or doze%
2244 \or treze%
2245 \or quatorze%
2246 \or quinze%
2247 \or dezesseis%
2248 \or dezessete%
2249 \or dezoito%
2250 \or dezenove%
2251 \fi
2252 }%
2253 \global\let\@@teenstringportuges\@@teenstringportuges

```

Hundreds:

```

2254 \newcommand*\@@hundredstringportuges[1]{%
2255 \ifcase#1\relax
2256 \or cento%
2257 \or duzentos%

```

```

2258 \or trezentos%
2259 \or quatrocentos%
2260 \or quinhentos%
2261 \or seiscentos%
2262 \or setecentos%
2263 \or oitocentos%
2264 \or novecentos%
2265 \fi
2266 }%
2267 \global\let\@@hundredstringportuges\@@hundredstringportuges

Hundreds (feminine):
2268 \newcommand*\@@hundredstringFportuges[1]{%
2269 \ifcase#1\relax
2270 \or cento%
2271 \or duzentas%
2272 \or trezentas%
2273 \or quatrocentas%
2274 \or quinhentas%
2275 \or seiscentas%
2276 \or setecentas%
2277 \or oitocentas%
2278 \or novecentas%
2279 \fi
2280 }%
2281 \global\let\@@hundredstringFportuges\@@hundredstringFportuges

Units (initial letter in upper case):
2282 \newcommand*\@@Unitstringportuges[1]{%
2283 \ifcase#1\relax
2284 Zero%
2285 \or Um%
2286 \or Dois%
2287 \or Tr\^es%
2288 \or Quatro%
2289 \or Cinco%
2290 \or Seis%
2291 \or Sete%
2292 \or Oito%
2293 \or Nove%
2294 \fi
2295 }%
2296 \global\let\@@Unitstringportuges\@@Unitstringportuges

As above, but feminine:
2297 \newcommand*\@@UnitstringFportuges[1]{%
2298 \ifcase#1\relax
2299 Zera%
2300 \or Uma%
2301 \or Duas%
2302 \or Tr\^es%

```

```

2303     \or Quatro%
2304     \or Cinco%
2305     \or Seis%
2306     \or Sete%
2307     \or Oito%
2308     \or Nove%
2309     \fi
2310 }%
2311 \global\let\@@UnitstringFportuges\@@UnitstringFportuges

    Tens (with initial letter in upper case):
2312 \newcommand*\@@Tenstringportuges[1]{%
2313     \ifcase#1\relax
2314         \or Dez%
2315         \or Vinte%
2316         \or Trinta%
2317         \or Quarenta%
2318         \or Cinq\ "uenta%
2319         \or Sessenta%
2320         \or Setenta%
2321         \or Oitenta%
2322         \or Noventa%
2323         \or Cem%
2324     \fi
2325 }%
2326 \global\let\@@Tenstringportuges\@@Tenstringportuges

    Teens (with initial letter in upper case):
2327 \newcommand*\@@Teenstringportuges[1]{%
2328     \ifcase#1\relax
2329         Dez%
2330         \or Onze%
2331         \or Doze%
2332         \or Treze%
2333         \or Quatorze%
2334         \or Quinze%
2335         \or Dezesesseis%
2336         \or Dezessete%
2337         \or Dezoito%
2338         \or Dezenove%
2339     \fi
2340 }%
2341 \global\let\@@Teenstringportuges\@@Teenstringportuges

    Hundreds (with initial letter in upper case):
2342 \newcommand*\@@Hundredstringportuges[1]{%
2343     \ifcase#1\relax
2344         \or Cento%
2345         \or Duzentos%
2346         \or Trezentos%
2347         \or Quatrocentos%

```

```

2348 \or Quinhentos%
2349 \or Seiscentos%
2350 \or Setecentos%
2351 \or Oitocentos%
2352 \or Novecentos%
2353 \fi
2354 }%
2355 \global\let\@@Hundredstringportuges\@@Hundredstringportuges

```

As above, but feminine:

```

2356 \newcommand*\@@HundredstringFportuges[1]{%
2357 \ifcase#1\relax
2358 \or Cento%
2359 \or Duzentas%
2360 \or Trezentas%
2361 \or Quatrocentas%
2362 \or Quinhentas%
2363 \or Seiscentas%
2364 \or Setecentas%
2365 \or Oitocentas%
2366 \or Novecentas%
2367 \fi
2368 }%
2369 \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.)

```

2370 \DeclareRobustCommand{\@numberstringMportuges}[2]{%
2371 \let\@unitstring=\@unitstringportuges
2372 \let\@teenstring=\@teenstringportuges
2373 \let\@tenstring=\@tenstringportuges
2374 \let\@hundredstring=\@hundredstringportuges
2375 \def\@hundred{cem}\def\@thousand{mil}%
2376 \def\@andname{e}%
2377 \@numberstringportuges{#1}{#2}%
2378 }%
2379 \global\let\@numberstringMportuges\@numberstringMportuges

```

As above, but feminine form:

```

2380 \DeclareRobustCommand{\@numberstringFportuges}[2]{%
2381 \let\@unitstring=\@unitstringFportuges
2382 \let\@teenstring=\@teenstringportuges
2383 \let\@tenstring=\@tenstringportuges
2384 \let\@hundredstring=\@hundredstringFportuges
2385 \def\@hundred{cem}\def\@thousand{mil}%
2386 \def\@andname{e}%
2387 \@numberstringportuges{#1}{#2}%
2388 }%

```

2389 \global\let\@numberstringFportuges\@numberstringFportuges

Make neuter same as masculine:

2390 \global\let\@numberstringNportuges\@numberstringMportuges

As above, but initial letters in upper case:

2391 \DeclareRobustCommand{\@NumberstringMportuges}[2]{%

2392 \let\@unitstring=\@Unitstringportuges

2393 \let\@teenstring=\@Teenstringportuges

2394 \let\@tenstring=\@Tenstringportuges

2395 \let\@hundredstring=\@Hundredstringportuges

2396 \def\@hundred{Cem}\def\@thousand{Mil}%

2397 \def\@andname{e}%

2398 \@numberstringportuges{#1}{#2}%

2399 }%

2400 \global\let\@NumberstringMportuges\@NumberstringMportuges

As above, but feminine form:

2401 \DeclareRobustCommand{\@NumberstringFportuges}[2]{%

2402 \let\@unitstring=\@UnitstringFportuges

2403 \let\@teenstring=\@Teenstringportuges

2404 \let\@tenstring=\@Tenstringportuges

2405 \let\@hundredstring=\@HundredstringFportuges

2406 \def\@hundred{Cem}\def\@thousand{Mil}%

2407 \def\@andname{e}%

2408 \@numberstringportuges{#1}{#2}%

2409 }%

2410 \global\let\@NumberstringFportuges\@NumberstringFportuges

Make neuter same as masculine:

2411 \global\let\@NumberstringNportuges\@NumberstringMportuges

As above, but for ordinals.

2412 \DeclareRobustCommand{\@OrdinalstringMportuges}[2]{%

2413 \let\@unitthstring=\@Unitthstringportuges

2414 \let\@unitstring=\@Unitstringportuges

2415 \let\@teenthstring=\@Teenthstringportuges

2416 \let\@tenthstring=\@Tenthstringportuges

2417 \let\@hundredthstring=\@Hundredthstringportuges

2418 \def\@thousandth{mil\ 'esimo}%

2419 \@ordinalstringportuges{#1}{#2}%

2420 }%

2421 \global\let\@OrdinalstringMportuges\@OrdinalstringMportuges

Feminine form:

2422 \DeclareRobustCommand{\@OrdinalstringFportuges}[2]{%

2423 \let\@unitthstring=\@UnitthstringFportuges

2424 \let\@unitstring=\@UnitstringFportuges

2425 \let\@teenthstring=\@Teenthstringportuges

2426 \let\@tenthstring=\@TenthstringFportuges

2427 \let\@hundredthstring=\@HundredthstringFportuges

2428 \def\@thousandth{mil\ 'esima}%

```

2429 \@ordinalstringportuges{#1}{#2}%
2430 }%
2431 \global\let\@ordinalstringFportuges\@ordinalstringFportuges
    Make neuter same as masculine:
2432 \global\let\@ordinalstringNportuges\@ordinalstringMportuges
    As above, but initial letters in upper case (masculine):
2433 \DeclareRobustCommand{\@OrdinalstringMportuges}[2]{%
2434   \let\@unitthstring=\@Unitthstringportuges
2435   \let\@unitstring=\@Unitstringportuges
2436   \let\@teenthstring=\@Teenthstringportuges
2437   \let\@tenthstring=\@Tenthstringportuges
2438   \let\@hundredthstring=\@Hundredthstringportuges
2439   \def\@thousandth{Mil\ 'esimo}%
2440   \@ordinalstringportuges{#1}{#2}%
2441 }%
2442 \global\let\@OrdinalstringMportuges\@OrdinalstringMportuges
    Feminine form:
2443 \DeclareRobustCommand{\@OrdinalstringFportuges}[2]{%
2444   \let\@unitthstring=\@UnitthstringFportuges
2445   \let\@unitstring=\@UnitstringFportuges
2446   \let\@teenthstring=\@TeenthstringFportuges
2447   \let\@tenthstring=\@TenthstringFportuges
2448   \let\@hundredthstring=\@HundredthstringFportuges
2449   \def\@thousandth{Mil\ 'esima}%
2450   \@ordinalstringportuges{#1}{#2}%
2451 }%
2452 \global\let\@OrdinalstringFportuges\@OrdinalstringFportuges
    Make neuter same as masculine:
2453 \global\let\@OrdinalstringNportuges\@OrdinalstringMportuges
    In order to do the ordinals, split into units, teens, tens and hundreds. Units:
2454 \newcommand*\@unitthstringportuges[1]{%
2455   \ifcase#1\relax
2456     zero%
2457     \or primeiro%
2458     \or segundo%
2459     \or terceiro%
2460     \or quarto%
2461     \or quinto%
2462     \or sexto%
2463     \or s\ 'etimo%
2464     \or oitavo%
2465     \or nono%
2466   \fi
2467 }%
2468 \global\let\@unitthstringportuges\@unitthstringportuges
    Tens:

```

```

2469 \newcommand*\@@tenthstringportuges[1]{%
2470   \ifcase#1\relax
2471     \or d\`ecimo%
2472     \or vig\`esimo%
2473     \or trig\`esimo%
2474     \or quadrag\`esimo%
2475     \or q\`uinquag\`esimo%
2476     \or sexag\`esimo%
2477     \or setuag\`esimo%
2478     \or octog\`esimo%
2479     \or nonag\`esimo%
2480   \fi
2481 }%
2482 \global\let\@@tenthstringportuges\@@tenthstringportuges

  Teens:
2483 \newcommand*\@@teenthstringportuges[1]{%
2484   \@tenthstring{1}%
2485   \ifnum#1>0\relax
2486     -\@unitthstring{#1}%
2487   \fi
2488 }%
2489 \global\let\@@teenthstringportuges\@@teenthstringportuges

  Hundreds:
2490 \newcommand*\@@hundredthstringportuges[1]{%
2491   \ifcase#1\relax
2492     \or cent\`esimo%
2493     \or ducent\`esimo%
2494     \or trecent\`esimo%
2495     \or quadringent\`esimo%
2496     \or q\`uingent\`esimo%
2497     \or seiscent\`esimo%
2498     \or setingent\`esimo%
2499     \or octingent\`esimo%
2500     \or nongent\`esimo%
2501   \fi
2502 }%
2503 \global\let\@@hundredthstringportuges\@@hundredthstringportuges

  Units (feminine):
2504 \newcommand*\@@unitthstringFportuges[1]{%
2505   \ifcase#1\relax
2506     zero%
2507     \or primeira%
2508     \or segunda%
2509     \or terceira%
2510     \or quarta%
2511     \or quinta%
2512     \or sexta%
2513     \or s\`etima%

```

```

2514 \or oitava%
2515 \or nona%
2516 \fi
2517 }%
2518 \global\let\@@unitthstringFportuges\@@unitthstringFportuges

```

Tens (feminine):

```

2519 \newcommand*\@@tenthstringFportuges[1]{%
2520 \ifcase#1\relax
2521 \or d\`ecima%
2522 \or vig\`esima%
2523 \or trig\`esima%
2524 \or quadrag\`esima%
2525 \or q\`uinquag\`esima%
2526 \or sexag\`esima%
2527 \or setuag\`esima%
2528 \or octog\`esima%
2529 \or nonag\`esima%
2530 \fi
2531 }%
2532 \global\let\@@tenthstringFportuges\@@tenthstringFportuges

```

Hundreds (feminine):

```

2533 \newcommand*\@@hundredthstringFportuges[1]{%
2534 \ifcase#1\relax
2535 \or cent\`esima%
2536 \or ducent\`esima%
2537 \or trecent\`esima%
2538 \or quadringent\`esima%
2539 \or q\`uingent\`esima%
2540 \or seiscent\`esima%
2541 \or setingent\`esima%
2542 \or octingent\`esima%
2543 \or nongent\`esima%
2544 \fi
2545 }%
2546 \global\let\@@hundredthstringFportuges\@@hundredthstringFportuges

```

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

```

2547 \newcommand*\@@Unitthstringportuges[1]{%
2548 \ifcase#1\relax
2549 Zero%
2550 \or Primeiro%
2551 \or Segundo%
2552 \or Terceiro%
2553 \or Quarto%
2554 \or Quinto%
2555 \or Sexto%
2556 \or S\`etimo%
2557 \or Oitavo%
2558 \or Nono%

```



```

2559 \fi
2560 }%
2561 \global\let\@@Unitthstringportuges\@@Unitthstringportuges

Tens:
2562 \newcommand*\@@Tenthstringportuges[1]{%
2563 \ifcase#1\relax
2564 \or D\`ecimo%
2565 \or Vig\`esimo%
2566 \or Trig\`esimo%
2567 \or Quadrag\`esimo%
2568 \or Q\`uinquag\`esimo%
2569 \or Sexag\`esimo%
2570 \or Setuag\`esimo%
2571 \or Octog\`esimo%
2572 \or Nonag\`esimo%
2573 \fi
2574 }%
2575 \global\let\@@Tenthstringportuges\@@Tenthstringportuges

Hundreds:
2576 \newcommand*\@@Hundredthstringportuges[1]{%
2577 \ifcase#1\relax
2578 \or Cent\`esimo%
2579 \or Ducent\`esimo%
2580 \or Trecent\`esimo%
2581 \or Quadringent\`esimo%
2582 \or Q\`uingent\`esimo%
2583 \or Seiscent\`esimo%
2584 \or Setingent\`esimo%
2585 \or Octingent\`esimo%
2586 \or Nongent\`esimo%
2587 \fi
2588 }%
2589 \global\let\@@Hundredthstringportuges\@@Hundredthstringportuges

As above, but feminine. Units:
2590 \newcommand*\@@UnitthstringFportuges[1]{%
2591 \ifcase#1\relax
2592 Zera%
2593 \or Primeira%
2594 \or Segunda%
2595 \or Terceira%
2596 \or Quarta%
2597 \or Quinta%
2598 \or Sexta%
2599 \or S\`etima%
2600 \or Oitava%
2601 \or Nona%
2602 \fi
2603 }%

```

2604 \global\let\@@UnitthstringFportuges\@@UnitthstringFportuges

Tens (feminine);

2605 \newcommand*\@@TenthstringFportuges[1]{%

2606 \ifcase#1\relax

2607 \or D\'ecima%

2608 \or Vig\'esima%

2609 \or Trig\'esima%

2610 \or Quadrag\'esima%

2611 \or Q\'uinquag\'esima%

2612 \or Sexag\'esima%

2613 \or Setuag\'esima%

2614 \or Octog\'esima%

2615 \or Nonag\'esima%

2616 \fi

2617 }%

2618 \global\let\@@TenthstringFportuges\@@TenthstringFportuges

Hundreds (feminine):

2619 \newcommand*\@@HundredthstringFportuges[1]{%

2620 \ifcase#1\relax

2621 \or Cent\'esima%

2622 \or Ducent\'esima%

2623 \or Trecent\'esima%

2624 \or Quadringent\'esima%

2625 \or Q\'uingent\'esima%

2626 \or Seiscent\'esima%

2627 \or Setingent\'esima%

2628 \or Octingent\'esima%

2629 \or Nongent\'esima%

2630 \fi

2631 }%

2632 \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.)

2633 \newcommand*\@@numberstringportuges[2]{%

2634 \ifnum#1>99999\relax

2635 \PackageError{fmtcount}{Out of range}%

2636 {This macro only works for values less than 100000}%

2637 \else

2638 \ifnum#1<0\relax

2639 \PackageError{fmtcount}{Negative numbers not permitted}%

2640 {This macro does not work for negative numbers, however

2641 you can try typing "minus" first, and then pass the modulus of

2642 this number}%

2643 \fi

2644 \fi

```

2645 \def#2{}%
2646 \@strctr=#1\relax \divide\@strctr by 1000\relax
2647 \ifnum\@strctr>9\relax
    #1 is greater or equal to 10000
2648 \divide\@strctr by 10\relax
2649 \ifnum\@strctr>1\relax
2650 \let\@fc@numstr#2\relax
2651 \protected@edef#2{\@fc@numstr\@tenstring{\@strctr}}%
2652 \@strctr=#1 \divide\@strctr by 1000\relax
2653 \@FCmodulo{\@strctr}{10}%
2654 \ifnum\@strctr>0
2655 \ifnum\@strctr=1\relax
2656 \let\@fc@numstr#2\relax
2657 \protected@edef#2{\@fc@numstr\ \@andname}%
2658 \fi
2659 \let\@fc@numstr#2\relax
2660 \protected@edef#2{\@fc@numstr\ \@unitstring{\@strctr}}%
2661 \fi
2662 \else
2663 \@strctr=#1\relax
2664 \divide\@strctr by 1000\relax
2665 \@FCmodulo{\@strctr}{10}%
2666 \let\@fc@numstr#2\relax
2667 \protected@edef#2{\@fc@numstr\@teenstring{\@strctr}}%
2668 \fi
2669 \let\@fc@numstr#2\relax
2670 \protected@edef#2{\@fc@numstr\ \@thousand}%
2671 \else
2672 \ifnum\@strctr>0\relax
2673 \ifnum\@strctr>1\relax
2674 \let\@fc@numstr#2\relax
2675 \protected@edef#2{\@fc@numstr\@unitstring{\@strctr}\ }%
2676 \fi
2677 \let\@fc@numstr#2\relax
2678 \protected@edef#2{\@fc@numstr\@thousand}%
2679 \fi
2680 \fi
2681 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
2682 \divide\@strctr by 100\relax
2683 \ifnum\@strctr>0\relax
2684 \ifnum#1>1000 \relax
2685 \let\@fc@numstr#2\relax
2686 \protected@edef#2{\@fc@numstr\ }%
2687 \fi
2688 \@tmpstrctr=#1\relax
2689 \@FCmodulo{\@tmpstrctr}{1000}%
2690 \let\@fc@numstr#2\relax
2691 \ifnum\@tmpstrctr=100\relax
2692 \protected@edef#2{\@fc@numstr\@tenstring{10}}%

```

```

2693 \else
2694 \protected@edef#2{\@fc@numstr\@hundredstring{\@strctr}}%
2695 \fi%
2696 \fi
2697 \@strctr=#1\relax \@FCmodulo{\@strctr}{100}%
2698 \ifnum#1>100\relax
2699 \ifnum\@strctr>0\relax
2700 \let\@fc@numstr#2\relax
2701 \protected@edef#2{\@fc@numstr\ \@andname\ }%
2702 \fi
2703 \fi
2704 \ifnum\@strctr>19\relax
2705 \divide\@strctr by 10\relax
2706 \let\@fc@numstr#2\relax
2707 \protected@edef#2{\@fc@numstr\@tenstring{\@strctr}}%
2708 \@strctr=#1\relax \@FCmodulo{\@strctr}{10}%
2709 \ifnum\@strctr>0
2710 \ifnum\@strctr=1\relax
2711 \let\@fc@numstr#2\relax
2712 \protected@edef#2{\@fc@numstr\ \@andname}%
2713 \else
2714 \ifnum#1>100\relax
2715 \let\@fc@numstr#2\relax
2716 \protected@edef#2{\@fc@numstr\ \@andname}%
2717 \fi
2718 \fi
2719 \let\@fc@numstr#2\relax
2720 \protected@edef#2{\@fc@numstr\ \@unitstring{\@strctr}}%
2721 \fi
2722 \else
2723 \ifnum\@strctr<10\relax
2724 \ifnum\@strctr=0\relax
2725 \ifnum#1<100\relax
2726 \let\@fc@numstr#2\relax
2727 \protected@edef#2{\@fc@numstr\@unitstring{\@strctr}}%
2728 \fi
2729 \else %(>0,<10)
2730 \let\@fc@numstr#2\relax
2731 \protected@edef#2{\@fc@numstr\@unitstring{\@strctr}}%
2732 \fi
2733 \else%>10
2734 \@FCmodulo{\@strctr}{10}%
2735 \let\@fc@numstr#2\relax
2736 \protected@edef#2{\@fc@numstr\@teenstring{\@strctr}}%
2737 \fi
2738 \fi
2739 }%
2740 \global\let\@numberstringportuges\@numberstringportuges

```

As above, but for ordinals.

```

2741 \newcommand*{\@@ordinalstringportuges[2]}{%
2742 \@strctr=#1\relax
2743 \ifnum#1>99999
2744 \PackageError{fmtcount}{Out of range}%
2745 {This macro only works for values less than 100000}%
2746 \else
2747 \ifnum#1<0
2748 \PackageError{fmtcount}{Negative numbers not permitted}%
2749 {This macro does not work for negative numbers, however
2750 you can try typing "minus" first, and then pass the modulus of
2751 this number}%
2752 \else
2753 \def#2{}%
2754 \ifnum\@strctr>999\relax
2755   \divide\@strctr by 1000\relax
2756   \ifnum\@strctr>1\relax
2757     \ifnum\@strctr>9\relax
2758       \@tmpstrctr=\@strctr
2759       \ifnum\@strctr<20
2760         \@FCmodulo{\@tmpstrctr}{10}%
2761         \let\@@fc@ordstr#2\relax
2762         \protected@edef#2{\@@fc@ordstr\@teenthstring{\@tmpstrctr}}%
2763       \else
2764         \divide\@tmpstrctr by 10\relax
2765         \let\@@fc@ordstr#2\relax
2766         \protected@edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
2767         \@tmpstrctr=\@strctr
2768         \@FCmodulo{\@tmpstrctr}{10}%
2769         \ifnum\@tmpstrctr>0\relax
2770           \let\@@fc@ordstr#2\relax
2771           \protected@edef#2{\@@fc@ordstr\@unitthstring{\@tmpstrctr}}%
2772         \fi
2773       \fi
2774     \else
2775       \let\@@fc@ordstr#2\relax
2776       \protected@edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
2777     \fi
2778   \fi
2779   \let\@@fc@ordstr#2\relax
2780   \protected@edef#2{\@@fc@ordstr\@thousandth}%
2781 \fi
2782 \@strctr=#1\relax
2783 \@FCmodulo{\@strctr}{1000}%
2784 \ifnum\@strctr>99\relax
2785   \@tmpstrctr=\@strctr
2786   \divide\@tmpstrctr by 100\relax
2787   \ifnum#1>1000\relax
2788     \let\@@fc@ordstr#2\relax
2789     \protected@edef#2{\@@fc@ordstr-}%

```

```

2790 \fi
2791 \let\@@fc@ordstr#2\relax
2792 \protected@edef#2{\@@fc@ordstr\@hundredthstring{\@tmpstrctr}}%
2793 \fi
2794 \@FCmodulo{\@strctr}{100}%
2795 \ifnum#1>99\relax
2796 \ifnum\@strctr>0\relax
2797 \let\@@fc@ordstr#2\relax
2798 \protected@edef#2{\@@fc@ordstr-}%
2799 \fi
2800 \fi
2801 \ifnum\@strctr>9\relax
2802 \@tmpstrctr=\@strctr
2803 \divide\@tmpstrctr by 10\relax
2804 \let\@@fc@ordstr#2\relax
2805 \protected@edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
2806 \@tmpstrctr=\@strctr
2807 \@FCmodulo{\@tmpstrctr}{10}%
2808 \ifnum\@tmpstrctr>0\relax
2809 \let\@@fc@ordstr#2\relax
2810 \protected@edef#2{\@@fc@ordstr-\@unitthstring{\@tmpstrctr}}%
2811 \fi
2812 \else
2813 \ifnum\@strctr=0\relax
2814 \ifnum#1=0\relax
2815 \let\@@fc@ordstr#2\relax
2816 \protected@edef#2{\@@fc@ordstr\@unitstring{0}}%
2817 \fi
2818 \else
2819 \let\@@fc@ordstr#2\relax
2820 \protected@edef#2{\@@fc@ordstr\@unitthstring{\@strctr}}%
2821 \fi
2822 \fi
2823 \fi
2824 \fi
2825 }%
2826 \global\let\@@ordinalstringportuges\@@ordinalstringportuges

```

9.0.13 fc-portuguese.def

```
2827 \ProvidesFCLanguage{portuguese}[2014/06/09]%
```

Load fc-portuges.def if not already loaded

```
2828 \FCloadlang{portuges}%
```

Set portuguese to be equivalent to portuges.

```
2829 \global\let\@ordinalMportuguese=\@ordinalMportuges
```

```
2830 \global\let\@ordinalFportuguese=\@ordinalFportuges
```

```
2831 \global\let\@ordinalNportuguese=\@ordinalNportuges
```

```
2832 \global\let\@numberstringMportuguese=\@numberstringMportuges
```

```

2833 \global\let \@numberstringFportuguese=\@numberstringFportuges
2834 \global\let \@numberstringNportuguese=\@numberstringNportuges
2835 \global\let \@NumberstringMportuguese=\@NumberstringMportuges
2836 \global\let \@NumberstringFportuguese=\@NumberstringFportuges
2837 \global\let \@NumberstringNportuguese=\@NumberstringNportuges
2838 \global\let \@ordinalstringMportuguese=\@ordinalstringMportuges
2839 \global\let \@ordinalstringFportuguese=\@ordinalstringFportuges
2840 \global\let \@ordinalstringNportuguese=\@ordinalstringNportuges
2841 \global\let \@OrdinalstringMportuguese=\@OrdinalstringMportuges
2842 \global\let \@OrdinalstringFportuguese=\@OrdinalstringFportuges
2843 \global\let \@OrdinalstringNportuguese=\@OrdinalstringNportuges

```

9.0.14 fc-spanish.def

Spanish definitions

```
2844 \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:

```

2845 \newcommand* \@ordinalMspanish[2]{%
2846   \edef#2{\number#1\relax\noexpand\fmtord{o}}%
2847 }%
2848 \global\let \@ordinalMspanish \@ordinalMspanish

```

Feminine:

```

2849 \newcommand* \@ordinalFspanish[2]{%
2850   \edef#2{\number#1\relax\noexpand\fmtord{a}}%
2851 }%
2852 \global\let \@ordinalFspanish \@ordinalFspanish

```

Make neuter same as masculine:

```
2853 \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):

```

2854 \newcommand* @@unitstringspanish[1]{%
2855   \ifcase#1\relax
2856     cero%
2857     \or uno%
2858     \or dos%
2859     \or tres%
2860     \or cuatro%
2861     \or cinco%
2862     \or seis%
2863     \or siete%
2864     \or ocho%
2865     \or nueve%
2866   \fi
2867 }%

```

2868 \global\let\@@unitstringspanish\@@unitstringspanish

Feminine:

2869 \newcommand*\@@unitstringFspanish[1]{%

2870 \ifcase#1\relax

2871 \cerca%

2872 \or una%

2873 \or dos%

2874 \or tres%

2875 \or cuatro%

2876 \or cinco%

2877 \or seis%

2878 \or siete%

2879 \or ocho%

2880 \or nueve%

2881 \fi

2882 }%

2883 \global\let\@@unitstringFspanish\@@unitstringFspanish

Tens (argument must go from 1 to 10):

2884 \newcommand*\@@tenstringspanish[1]{%

2885 \ifcase#1\relax

2886 \or diez%

2887 \or veinte%

2888 \or treinta%

2889 \or cuarenta%

2890 \or cincuenta%

2891 \or sesenta%

2892 \or setenta%

2893 \or ochenta%

2894 \or noventa%

2895 \or cien%

2896 \fi

2897 }%

2898 \global\let\@@tenstringspanish\@@tenstringspanish

Teens:

2899 \newcommand*\@@teenstringspanish[1]{%

2900 \ifcase#1\relax

2901 \diez%

2902 \or once%

2903 \or doce%

2904 \or trece%

2905 \or catorce%

2906 \or quince%

2907 \or diecis\'eis%

2908 \or diecisiete%

2909 \or dieciocho%

2910 \or diecinueve%

2911 \fi

2912 }%

2913 \global\let\@@teenstringspanish\@@teenstringspanish

Twenties:

2914 \newcommand*\@@twentystringspanish[1]{%

2915 \ifcase#1\relax

2916 veinte%

2917 \or veintiuno%

2918 \or veintid\'os%

2919 \or veintitr\'es%

2920 \or veinticuatro%

2921 \or veinticinco%

2922 \or veintis\'eis%

2923 \or veintisiete%

2924 \or veintiocho%

2925 \or veintinueve%

2926 \fi

2927 }%

2928 \global\let\@@twentystringspanish\@@twentystringspanish

Feminine form:

2929 \newcommand*\@@twentystringFspanish[1]{%

2930 \ifcase#1\relax

2931 veinte%

2932 \or veintiuna%

2933 \or veintid\'os%

2934 \or veintitr\'es%

2935 \or veinticuatro%

2936 \or veinticinco%

2937 \or veintis\'eis%

2938 \or veintisiete%

2939 \or veintiocho%

2940 \or veintinueve%

2941 \fi

2942 }%

2943 \global\let\@@twentystringFspanish\@@twentystringFspanish

Hundreds:

2944 \newcommand*\@@hundredstringspanish[1]{%

2945 \ifcase#1\relax

2946 \or ciento%

2947 \or doscientos%

2948 \or trescientos%

2949 \or cuatrocientos%

2950 \or quinientos%

2951 \or seiscientos%

2952 \or setecientos%

2953 \or ochocientos%

2954 \or novecientos%

2955 \fi

2956 }%

2957 \global\let\@@hundredstringspanish\@@hundredstringspanish

Feminine form:

```
2958 \newcommand*{\@@hundredstringFspanish[1]}{%
2959   \ifcase#1\relax
2960     \or cienta%
2961     \or doscientas%
2962     \or trescientas%
2963     \or cuatrocientas%
2964     \or quinientas%
2965     \or seiscientas%
2966     \or setecientas%
2967     \or ochocientas%
2968     \or novecientas%
2969   \fi
2970 }%
2971 \global\let\@@hundredstringFspanish\@@hundredstringFspanish
```

As above, but with initial letter uppercase:

```
2972 \newcommand*{\@@Unitstringspanish[1]}{%
2973   \ifcase#1\relax
2974     Cero%
2975     \or Uno%
2976     \or Dos%
2977     \or Tres%
2978     \or Cuatro%
2979     \or Cinco%
2980     \or Seis%
2981     \or Siete%
2982     \or Ocho%
2983     \or Nueve%
2984   \fi
2985 }%
2986 \global\let\@@Unitstringspanish\@@Unitstringspanish
```

Feminine form:

```
2987 \newcommand*{\@@UnitstringFspanish[1]}{%
2988   \ifcase#1\relax
2989     Cera%
2990     \or Una%
2991     \or Dos%
2992     \or Tres%
2993     \or Cuatro%
2994     \or Cinco%
2995     \or Seis%
2996     \or Siete%
2997     \or Ocho%
2998     \or Nueve%
2999   \fi
3000 }%
3001 \global\let\@@UnitstringFspanish\@@UnitstringFspanish
```

Tens:

```

3002%\changes{2.0}{2012-06-18}{fixed spelling mistake (correction
3003%provided by Fernando Maldonado)}
3004 \newcommand*\@@Tenstringspanish[1]{%
3005   \ifcase#1\relax
3006     \or Diez%
3007     \or Veinte%
3008     \or Treinta%
3009     \or Cuarenta%
3010     \or Cincuenta%
3011     \or Sesenta%
3012     \or Setenta%
3013     \or Ochenta%
3014     \or Noventa%
3015     \or Cien%
3016   \fi
3017 }%
3018 \global\let\@@Tenstringspanish\@@Tenstringspanish

```

Teens:

```

3019 \newcommand*\@@Teenstringspanish[1]{%
3020   \ifcase#1\relax
3021     Diez%
3022     \or Once%
3023     \or Doce%
3024     \or Trece%
3025     \or Catorce%
3026     \or Quince%
3027     \or Diecis\'eis%
3028     \or Diecisiete%
3029     \or Dieciocho%
3030     \or Diecinueve%
3031   \fi
3032 }%
3033 \global\let\@@Teenstringspanish\@@Teenstringspanish

```

Twenties:

```

3034 \newcommand*\@@Twentystingspanish[1]{%
3035   \ifcase#1\relax
3036     Veinte%
3037     \or Veintiuno%
3038     \or Veintid\'os%
3039     \or Veintitr\'es%
3040     \or Veinticuatro%
3041     \or Veinticinco%
3042     \or Veintis\'eis%
3043     \or Veintisiete%
3044     \or Veintiocho%
3045     \or Veintinueve%
3046   \fi
3047 }%

```

3048 \global\let\@@Twentystringspanish\@@Twentystringspanish

Feminine form:

3049 \newcommand*\@@TwentystringFspanish[1]{%

3050 \ifcase#1\relax

3051 Veinte%

3052 \or Veintiuna%

3053 \or Veintid\'os%

3054 \or Veintitr\'es%

3055 \or Veinticuatro%

3056 \or Veinticinco%

3057 \or Veintis\'eis%

3058 \or Veintisiete%

3059 \or Veintiocho%

3060 \or Veintinueve%

3061 \fi

3062 }%

3063 \global\let\@@TwentystringFspanish\@@TwentystringFspanish

Hundreds:

3064 \newcommand*\@@Hundredstringspanish[1]{%

3065 \ifcase#1\relax

3066 \or Ciento%

3067 \or Doscientos%

3068 \or Trescientos%

3069 \or Cuatrocientos%

3070 \or Quinientos%

3071 \or Seiscientos%

3072 \or Setecientos%

3073 \or Ochocientos%

3074 \or Novecientos%

3075 \fi

3076 }%

3077 \global\let\@@Hundredstringspanish\@@Hundredstringspanish

Feminine form:

3078 \newcommand*\@@HundredstringFspanish[1]{%

3079 \ifcase#1\relax

3080 \or Cienta%

3081 \or Doscientas%

3082 \or Trescientas%

3083 \or Cuatrocientas%

3084 \or Quinientas%

3085 \or Seiscientas%

3086 \or Setecientas%

3087 \or Ochocientas%

3088 \or Novecientas%

3089 \fi

3090 }%

3091 \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.)

```

3092 \DeclareRobustCommand{\@numberstringMspanish}[2]{%
3093   \let\@unitstring=\@unitstringspanish
3094   \let\@teenstring=\@teenstringspanish
3095   \let\@tenstring=\@tenstringspanish
3096   \let\@twentystring=\@twentystringspanish
3097   \let\@hundredstring=\@hundredstringspanish
3098   \def\@hundred{cien}\def\@thousand{mil}%
3099   \def\@andname{y}%
3100   \@numberstringspanish{#1}{#2}%
3101 }%
3102 \global\let\@numberstringMspanish\@numberstringMspanish

```

Feminine form:

```

3103 \DeclareRobustCommand{\@numberstringFspanish}[2]{%
3104   \let\@unitstring=\@unitstringFspanish
3105   \let\@teenstring=\@teenstringspanish
3106   \let\@tenstring=\@tenstringspanish
3107   \let\@twentystring=\@twentystringFspanish
3108   \let\@hundredstring=\@hundredstringFspanish
3109   \def\@hundred{cien}\def\@thousand{mil}%
3110   \def\@andname{b}%
3111   \@numberstringspanish{#1}{#2}%
3112 }%
3113 \global\let\@numberstringFspanish\@numberstringFspanish

```

Make neuter same as masculine:

```

3114 \global\let\@numberstringNspanish\@numberstringMspanish

```

As above, but initial letters in upper case:

```

3115 \DeclareRobustCommand{\@NumberstringMspanish}[2]{%
3116   \let\@unitstring=\@Unitstringspanish
3117   \let\@teenstring=\@Teenstringspanish
3118   \let\@tenstring=\@Tenstringspanish
3119   \let\@twentystring=\@Twentystringspanish
3120   \let\@hundredstring=\@Hundredstringspanish
3121   \def\@andname{y}%
3122   \def\@hundred{Cien}\def\@thousand{Mil}%
3123   \@numberstringspanish{#1}{#2}%
3124 }%
3125 \global\let\@NumberstringMspanish\@NumberstringMspanish

```

Feminine form:

```

3126 \DeclareRobustCommand{\@NumberstringFspanish}[2]{%
3127   \let\@unitstring=\@UnitstringFspanish
3128   \let\@teenstring=\@Teenstringspanish
3129   \let\@tenstring=\@Tenstringspanish

```

```

3130 \let\@twentystring=\@TwentystringFspanish
3131 \let\@hundredstring=\@HundredstringFspanish
3132 \def\@andname{b}%
3133 \def\@hundred{Cien}\def\@thousand{Mil}%
3134 \@@numberstringspanish{#1}{#2}%
3135 }%
3136 \global\let\@NumberstringFspanish\@NumberstringFspanish

  Make neuter same as masculine:
3137 \global\let\@NumberstringNspanish\@NumberstringMspanish

  As above, but for ordinals.
3138 \DeclareRobustCommand{\@ordinalstringMspanish}[2]{%
3139 \let\@unitthstring=\@unitthstringspanish
3140 \let\@unitstring=\@unitstringspanish
3141 \let\@teenthstring=\@teenthstringspanish
3142 \let\@tenthstring=\@tenthstringspanish
3143 \let\@hundredthstring=\@hundredthstringspanish
3144 \def\@thousandth{mil\'esimo}%
3145 \@@ordinalstringspanish{#1}{#2}%
3146 }%
3147 \global\let\@ordinalstringMspanish\@ordinalstringMspanish

  Feminine form:
3148 \DeclareRobustCommand{\@ordinalstringFspanish}[2]{%
3149 \let\@unitthstring=\@unitthstringFspanish
3150 \let\@unitstring=\@unitstringFspanish
3151 \let\@teenthstring=\@teenthstringFspanish
3152 \let\@tenthstring=\@tenthstringFspanish
3153 \let\@hundredthstring=\@hundredthstringFspanish
3154 \def\@thousandth{mil\'esima}%
3155 \@@ordinalstringspanish{#1}{#2}%
3156 }%
3157 \global\let\@ordinalstringFspanish\@ordinalstringFspanish

  Make neuter same as masculine:
3158 \global\let\@ordinalstringNspanish\@ordinalstringMspanish

  As above, but with initial letters in upper case.
3159 \DeclareRobustCommand{\@OrdinalstringMspanish}[2]{%
3160 \let\@unitthstring=\@Unitthstringspanish
3161 \let\@unitstring=\@Unitstringspanish
3162 \let\@teenthstring=\@Teenthstringspanish
3163 \let\@tenthstring=\@Tenthstringspanish
3164 \let\@hundredthstring=\@Hundredthstringspanish
3165 \def\@thousandth{Mil\'esimo}%
3166 \@@ordinalstringspanish{#1}{#2}%
3167 }
3168 \global\let\@OrdinalstringMspanish\@OrdinalstringMspanish

  Feminine form:
3169 \DeclareRobustCommand{\@OrdinalstringFspanish}[2]{%

```

```

3170 \let\@unitthstring=\@UnitthstringFspanish
3171 \let\@unitstring=\@UnitstringFspanish
3172 \let\@teenthstring=\@TeenthstringFspanish
3173 \let\@tenthstring=\@TenthstringFspanish
3174 \let\@hundredthstring=\@HundredthstringFspanish
3175 \def\@thousandth{Mil\'esima}%
3176 \@ordinalstringspanish{#1}{#2}%
3177 }%
3178 \global\let\@OrdinalstringFspanish\@OrdinalstringFspanish

```

Make neuter same as masculine:

```

3179 \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:

```

3180 \newcommand*\@unitthstringspanish[1]{%
3181   \ifcase#1\relax
3182     cero%
3183     \or primero%
3184     \or segundo%
3185     \or tercero%
3186     \or cuarto%
3187     \or quinto%
3188     \or sexto%
3189     \or s\'eptimo%
3190     \or octavo%
3191     \or noveno%
3192   \fi
3193 }%
3194 \global\let\@unitthstringspanish\@unitthstringspanish

```

Tens:

```

3195 \newcommand*\@tenthstringspanish[1]{%
3196   \ifcase#1\relax
3197     \or d\'ecimo%
3198     \or vig\'esimo%
3199     \or trig\'esimo%
3200     \or cuadrag\'esimo%
3201     \or quincuag\'esimo%
3202     \or sexag\'esimo%
3203     \or septuag\'esimo%
3204     \or octog\'esimo%
3205     \or nonag\'esimo%
3206   \fi
3207 }%
3208 \global\let\@tenthstringspanish\@tenthstringspanish

```

Teens:

```

3209 \newcommand*\@teenthstringspanish[1]{%
3210   \ifcase#1\relax
3211     d\'ecimo%

```

```

3212 \or und\'ecimo%
3213 \or duod\'ecimo%
3214 \or decimotercero%
3215 \or decimocuarto%
3216 \or decimoquinto%
3217 \or decimosexto%
3218 \or decimos\'eptimo%
3219 \or decimoctavo%
3220 \or decimonoveno%
3221 \fi
3222 }%
3223 \global\let\@teenthstringspanish\@teenthstringspanish

  Hundreds:
3224 \newcommand*\@hundredthstringspanish[1]{%
3225 \ifcase#1\relax
3226 \or cent\'esimo%
3227 \or ducent\'esimo%
3228 \or tricent\'esimo%
3229 \or cuadringent\'esimo%
3230 \or quingent\'esimo%
3231 \or sexcent\'esimo%
3232 \or septing\'esimo%
3233 \or octingent\'esimo%
3234 \or noningent\'esimo%
3235 \fi
3236 }%
3237 \global\let\@hundredthstringspanish\@hundredthstringspanish

  Units (feminine):
3238 \newcommand*\@unitthstringFspanish[1]{%
3239 \ifcase#1\relax
3240 cera%
3241 \or primera%
3242 \or segunda%
3243 \or tercera%
3244 \or cuarta%
3245 \or quinta%
3246 \or sexta%
3247 \or s\'eptima%
3248 \or octava%
3249 \or novena%
3250 \fi
3251 }%
3252 \global\let\@unitthstringFspanish\@unitthstringFspanish

  Tens (feminine):
3253 \newcommand*\@tenthstringFspanish[1]{%
3254 \ifcase#1\relax
3255 \or d\'esima%
3256 \or vig\'esima%

```



```

3257 \or trig\'esima%
3258 \or cuadrag\'esima%
3259 \or quincuag\'esima%
3260 \or sexag\'esima%
3261 \or septuag\'esima%
3262 \or octog\'esima%
3263 \or nonag\'esima%
3264 \fi
3265 }%
3266 \global\let\@@tenthstringFspanish\@@tenthstringFspanish

    Teens (feminine)
3267 \newcommand*\@@teenthstringFspanish[1]{%
3268 \ifcase#1\relax
3269 d\'ecima%
3270 \or und\'ecima%
3271 \or duod\'ecima%
3272 \or decimotercera%
3273 \or decimocuarta%
3274 \or decimoquinta%
3275 \or decimosexta%
3276 \or decimos\'eptima%
3277 \or decimoctava%
3278 \or decimonovena%
3279 \fi
3280 }%
3281 \global\let\@@teenthstringFspanish\@@teenthstringFspanish

    Hundreds (feminine)
3282 \newcommand*\@@hundredthstringFspanish[1]{%
3283 \ifcase#1\relax
3284 \or cent\'esima%
3285 \or ducent\'esima%
3286 \or tricent\'esima%
3287 \or cuadringent\'esima%
3288 \or quingent\'esima%
3289 \or sexcent\'esima%
3290 \or septing\'esima%
3291 \or octingent\'esima%
3292 \or noningent\'esima%
3293 \fi
3294 }%
3295 \global\let\@@hundredthstringFspanish\@@hundredthstringFspanish

    As above, but with initial letters in upper case
3296 \newcommand*\@@Unitthstringspanish[1]{%
3297 \ifcase#1\relax
3298 Cero%
3299 \or Primero%
3300 \or Segundo%
3301 \or Tercero%

```

```

3302 \or Cuarto%
3303 \or Quinto%
3304 \or Sexto%
3305 \or S\'eptimo%
3306 \or Octavo%
3307 \or Noveno%
3308 \fi
3309 }%
3310 \global\let\@@Unitthstringspanish\@@Unitthstringspanish

```

Tens:

```

3311 \newcommand*\@@Tenthstringspanish[1]{%
3312 \ifcase#1\relax
3313 \or D\'ecimo%
3314 \or Vig\'esimo%
3315 \or Trig\'esimo%
3316 \or Cuadrag\'esimo%
3317 \or Quincuag\'esimo%
3318 \or Sexag\'esimo%
3319 \or Septuag\'esimo%
3320 \or Octog\'esimo%
3321 \or Nonag\'esimo%
3322 \fi
3323 }%
3324 \global\let\@@Tenthstringspanish\@@Tenthstringspanish

```

Teens:

```

3325 \newcommand*\@@Teenthstringspanish[1]{%
3326 \ifcase#1\relax
3327 D\'ecimo%
3328 \or Und\'ecimo%
3329 \or Duod\'ecimo%
3330 \or Decimotercero%
3331 \or Decimocuarto%
3332 \or Decimoquinto%
3333 \or Decimosexto%
3334 \or Decimos\'eptimo%
3335 \or Decimoctavo%
3336 \or Decimonoveno%
3337 \fi
3338 }%
3339 \global\let\@@Teenthstringspanish\@@Teenthstringspanish

```

Hundreds

```

3340 \newcommand*\@@Hundredthstringspanish[1]{%
3341 \ifcase#1\relax
3342 \or Cent\'esimo%
3343 \or Ducent\'esimo%
3344 \or Tricent\'esimo%
3345 \or Cuadringent\'esimo%
3346 \or Quingent\'esimo%

```

```

3347 \or Sexcent\'esimo%
3348 \or Septing\'esimo%
3349 \or Octingent\'esimo%
3350 \or Noningent\'esimo%
3351 \fi
3352 }%
3353 \global\let\@@Hundredthstringspanish\@@Hundredthstringspanish

```

As above, but feminine.

```

3354 \newcommand*\@@UnitthstringFspanish[1]{%
3355 \ifcase#1\relax
3356 Cera%
3357 \or Primera%
3358 \or Segunda%
3359 \or Tercera%
3360 \or Cuarta%
3361 \or Quinta%
3362 \or Sexta%
3363 \or S\'eptima%
3364 \or Octava%
3365 \or Novena%
3366 \fi
3367 }%
3368 \global\let\@@UnitthstringFspanish\@@UnitthstringFspanish

```

Tens (feminine)

```

3369 \newcommand*\@@TenthstringFspanish[1]{%
3370 \ifcase#1\relax
3371 \or D\'ecima%
3372 \or Vig\'esima%
3373 \or Trig\'esima%
3374 \or Cuadrag\'esima%
3375 \or Quincuag\'esima%
3376 \or Sexag\'esima%
3377 \or Septuag\'esima%
3378 \or Octog\'esima%
3379 \or Nonag\'esima%
3380 \fi
3381 }%
3382 \global\let\@@TenthstringFspanish\@@TenthstringFspanish

```

Teens (feminine):

```

3383 \newcommand*\@@TeenthstringFspanish[1]{%
3384 \ifcase#1\relax
3385 D\'ecima%
3386 \or Und\'ecima%
3387 \or Duod\'ecima%
3388 \or Decimotercera%
3389 \or Decimocuarta%
3390 \or Decimoquinta%
3391 \or Decimosexta%

```

```

3392 \or Decimos\'eptima%
3393 \or Decimoctava%
3394 \or Decimonovena%
3395 \fi
3396 }%
3397 \global\let\@@TeenthstringFspanish\@@TeenthstringFspanish

```

Hundreds (feminine):

```

3398 \newcommand*\@@HundredthstringFspanish[1]{%
3399 \ifcase#1\relax
3400 \or Cent\'esima%
3401 \or Ducent\'esima%
3402 \or Tricent\'esima%
3403 \or Cuadringent\'esima%
3404 \or Quingent\'esima%
3405 \or Sexcent\'esima%
3406 \or Septing\'esima%
3407 \or Octingent\'esima%
3408 \or Noningent\'esima%
3409 \fi
3410 }%
3411 \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.)

```

3412 \newcommand*\@@numberstringspanish[2]{%
3413 \ifnum#1>99999
3414 \PackageError{fmtcount}{Out of range}%
3415 {This macro only works for values less than 100000}%
3416 \else
3417 \ifnum#1<0
3418 \PackageError{fmtcount}{Negative numbers not permitted}%
3419 {This macro does not work for negative numbers, however
3420 you can try typing "minus" first, and then pass the modulus of
3421 this number}%
3422 \fi
3423 \fi
3424 \def#2{}%
3425 \@strctr=#1\relax \divide\@strctr by 1000\relax
3426 \ifnum\@strctr>9
3427 \divide\@strctr by 10
3428 \ifnum\@strctr>1
3429 \let\@@fc@numstr#2\relax
3430 \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
3431 \@strctr=#1 \divide\@strctr by 1000\relax
3432 \@FCmodulo{\@strctr}{10}%

```

```

3433 \ifnum \@strctr>0\relax
3434 \let\@@fc@numstr#2\relax
3435 \edef#2{\@@fc@numstr\ \@andname\ \@unitstring{\@strctr}}%
3436 \fi
3437 \else
3438 \@strctr=#1\relax
3439 \divide\@strctr by 1000\relax
3440 \@FCmodulo{\@strctr}{10}%
3441 \let\@@fc@numstr#2\relax
3442 \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
3443 \fi
3444 \let\@@fc@numstr#2\relax
3445 \edef#2{\@@fc@numstr\ \@thousand}%
3446 \else
3447 \ifnum \@strctr>0\relax
3448 \ifnum \@strctr>1\relax
3449 \let\@@fc@numstr#2\relax
3450 \edef#2{\@@fc@numstr\@unitstring{\@strctr}\ }%
3451 \fi
3452 \let\@@fc@numstr#2\relax
3453 \edef#2{\@@fc@numstr\@thousand}%
3454 \fi
3455 \fi
3456 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
3457 \divide\@strctr by 100\relax
3458 \ifnum \@strctr>0\relax
3459 \ifnum#1>1000\relax
3460 \let\@@fc@numstr#2\relax
3461 \edef#2{\@@fc@numstr\ }%
3462 \fi
3463 \@tmpstrctr=#1\relax
3464 \@FCmodulo{\@tmpstrctr}{1000}%
3465 \ifnum \@tmpstrctr=100\relax
3466 \let\@@fc@numstr#2\relax
3467 \edef#2{\@@fc@numstr\@tenstring{10}}%
3468 \else
3469 \let\@@fc@numstr#2\relax
3470 \edef#2{\@@fc@numstr\@hundredstring{\@strctr}}%
3471 \fi
3472 \fi
3473 \@strctr=#1\relax \@FCmodulo{\@strctr}{100}%
3474 \ifnum#1>100\relax
3475 \ifnum \@strctr>0\relax
3476 \let\@@fc@numstr#2\relax
3477 \edef#2{\@@fc@numstr\ }%
3478 \fi
3479 \fi
3480 \ifnum \@strctr>29\relax
3481 \divide\@strctr by 10\relax

```

```

3482 \let\@fc@numstr#2\relax
3483 \edef#2{\@fc@numstr\@tenstring{\@strctr}}%
3484 \@strctr=#1\relax \@FCmodulo{\@strctr}{10}%
3485 \ifnum\@strctr>0\relax
3486 \let\@fc@numstr#2\relax
3487 \edef#2{\@fc@numstr\@andname\@unitstring{\@strctr}}%
3488 \fi
3489 \else
3490 \ifnum\@strctr<10\relax
3491 \ifnum\@strctr=0\relax
3492 \ifnum#1<100\relax
3493 \let\@fc@numstr#2\relax
3494 \edef#2{\@fc@numstr\@unitstring{\@strctr}}%
3495 \fi
3496 \else
3497 \let\@fc@numstr#2\relax
3498 \edef#2{\@fc@numstr\@unitstring{\@strctr}}%
3499 \fi
3500 \else
3501 \ifnum\@strctr>19\relax
3502 \@FCmodulo{\@strctr}{10}%
3503 \let\@fc@numstr#2\relax
3504 \edef#2{\@fc@numstr\@twentystring{\@strctr}}%
3505 \else
3506 \@FCmodulo{\@strctr}{10}%
3507 \let\@fc@numstr#2\relax
3508 \edef#2{\@fc@numstr\@teenstring{\@strctr}}%
3509 \fi
3510 \fi
3511 \fi
3512 }%
3513 \global\let\@numberstringspanish\@numberstringspanish

```

As above, but for ordinals

```

3514 \newcommand*\@ordinalstringspanish[2]{%
3515 \@strctr=#1\relax
3516 \ifnum#1>99999
3517 \PackageError{fmtcount}{Out of range}%
3518 {This macro only works for values less than 100000}%
3519 \else
3520 \ifnum#1<0
3521 \PackageError{fmtcount}{Negative numbers not permitted}%
3522 {This macro does not work for negative numbers, however
3523 you can try typing "minus" first, and then pass the modulus of
3524 this number}%
3525 \else
3526 \def#2{}%
3527 \ifnum\@strctr>999\relax
3528 \divide\@strctr by 1000\relax
3529 \ifnum\@strctr>1\relax

```

```

3530 \ifnum \@strctr>9\relax
3531 \@tmpstrctr=\@strctr
3532 \ifnum \@strctr<20
3533 \@FCmodulo{\@tmpstrctr}{10}%
3534 \let\@@fc@ordstr#2\relax
3535 \edef#2{\@@fc@ordstr\@teenthstring{\@tmpstrctr}}%
3536 \else
3537 \divide\@tmpstrctr by 10\relax
3538 \let\@@fc@ordstr#2\relax
3539 \edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
3540 \@tmpstrctr=\@strctr
3541 \@FCmodulo{\@tmpstrctr}{10}%
3542 \ifnum \@tmpstrctr>0\relax
3543 \let\@@fc@ordstr#2\relax
3544 \edef#2{\@@fc@ordstr\@unitthstring{\@tmpstrctr}}%
3545 \fi
3546 \fi
3547 \else
3548 \let\@@fc@ordstr#2\relax
3549 \edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
3550 \fi
3551 \fi
3552 \let\@@fc@ordstr#2\relax
3553 \edef#2{\@@fc@ordstr\@thousandth}%
3554 \fi
3555 \@strctr=#1\relax
3556 \@FCmodulo{\@strctr}{1000}%
3557 \ifnum \@strctr>99\relax
3558 \@tmpstrctr=\@strctr
3559 \divide\@tmpstrctr by 100\relax
3560 \ifnum#1>1000\relax
3561 \let\@@fc@ordstr#2\relax
3562 \edef#2{\@@fc@ordstr\ }%
3563 \fi
3564 \let\@@fc@ordstr#2\relax
3565 \edef#2{\@@fc@ordstr\@hundredthstring{\@tmpstrctr}}%
3566 \fi
3567 \@FCmodulo{\@strctr}{100}%
3568 \ifnum#1>99\relax
3569 \ifnum \@strctr>0\relax
3570 \let\@@fc@ordstr#2\relax
3571 \edef#2{\@@fc@ordstr\ }%
3572 \fi
3573 \fi
3574 \ifnum \@strctr>19\relax
3575 \@tmpstrctr=\@strctr
3576 \divide\@tmpstrctr by 10\relax
3577 \let\@@fc@ordstr#2\relax
3578 \edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%

```

```

3579 \@tmpstrctr=\@strctr
3580 \@FCmodulo{\@tmpstrctr}{10}%
3581 \ifnum \@tmpstrctr>0\relax
3582   \let\@fc@ordstr#2\relax
3583   \edef#2{\@fc@ordstr\ \@unitthstring{\@tmpstrctr}}%
3584 \fi
3585 \else
3586 \ifnum \@strctr>9\relax
3587   \@FCmodulo{\@strctr}{10}%
3588   \let\@fc@ordstr#2\relax
3589   \edef#2{\@fc@ordstr\@teenthstring{\@strctr}}%
3590 \else
3591 \ifnum \@strctr=0\relax
3592   \ifnum#1=0\relax
3593     \let\@fc@ordstr#2\relax
3594     \edef#2{\@fc@ordstr\@unitstring{0}}%
3595   \fi
3596 \else
3597   \let\@fc@ordstr#2\relax
3598   \edef#2{\@fc@ordstr\@unitthstring{\@strctr}}%
3599 \fi
3600 \fi
3601 \fi
3602 \fi
3603 \fi
3604 }%
3605 \global\let\@ordinalstringspanish\@ordinalstringspanish

```

9.0.15 fc-UKenglish.def

English definitions

```
3606 \ProvidesFCLanguage{UKenglish}[2013/08/17]%
```

Loaded fc-english.def if not already loaded

```
3607 \FCloadlang{english}%
```

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

```

3608 \global\let\@ordinalMUKenglish\@ordinalMenglish
3609 \global\let\@ordinalFUKenglish\@ordinalMenglish
3610 \global\let\@ordinalNUKenglish\@ordinalMenglish
3611 \global\let\@numberstringMUKenglish\@numberstringMenglish
3612 \global\let\@numberstringFUKenglish\@numberstringMenglish
3613 \global\let\@numberstringNUKenglish\@numberstringMenglish
3614 \global\let\@NumberstringMUKenglish\@NumberstringMenglish
3615 \global\let\@NumberstringFUKenglish\@NumberstringMenglish
3616 \global\let\@NumberstringNUKenglish\@NumberstringMenglish
3617 \global\let\@ordinalstringMUKenglish\@ordinalstringMenglish
3618 \global\let\@ordinalstringFUKenglish\@ordinalstringMenglish
3619 \global\let\@ordinalstringNUKenglish\@ordinalstringMenglish
3620 \global\let\@OrdinalstringMUKenglish\@OrdinalstringMenglish

```


3621 \global\let\@OrdinalstringFUKenglish\@OrdinalstringMenglish
 3622 \global\let\@OrdinalstringNUKenglish\@OrdinalstringMenglish

9.0.16 fc-USenglish.def

US English definitions

3623 \ProvidesFCLanguage{USenglish}[2013/08/17]%

Loaded fc-english.def if not already loaded

3624 \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.)

3625 \global\let\@ordinalMUSenglish\@ordinalMenglish

3626 \global\let\@ordinalFUSenglish\@ordinalMenglish

3627 \global\let\@ordinalNUSenglish\@ordinalMenglish

3628 \global\let\@numberstringMUSenglish\@numberstringMenglish

3629 \global\let\@numberstringFUSenglish\@numberstringMenglish

3630 \global\let\@numberstringNUSenglish\@numberstringMenglish

3631 \global\let\@NumberstringMUSenglish\@NumberstringMenglish

3632 \global\let\@NumberstringFUSenglish\@NumberstringMenglish

3633 \global\let\@NumberstringNUSenglish\@NumberstringMenglish

3634 \global\let\@ordinalstringMUSenglish\@ordinalstringMenglish

3635 \global\let\@ordinalstringFUSenglish\@ordinalstringMenglish

3636 \global\let\@ordinalstringNUSenglish\@ordinalstringMenglish

3637 \global\let\@OrdinalstringMUSenglish\@OrdinalstringMenglish

3638 \global\let\@OrdinalstringFUSenglish\@OrdinalstringMenglish

3639 \global\let\@OrdinalstringNUSenglish\@OrdinalstringMenglish

9.1 fcnumparser.sty

3640 \NeedsTeXFormat{LaTeX2e}

3641 \ProvidesPackage{fcnumparser}[2012/09/28]

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

3642 \def\fc@counter@parser#1{%

3643 \expandafter\fc@number@parser\expandafter{\the#1.}%

3644 }

3645 \newcount\fc@digit@counter

3646

3647 \def\fc@end@\fc@end}

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

3648 \def\fc@number@analysis#1\fc@nil{%

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

3649 \def\@tempb##1.##2\fc@nil{\def\fc@integer@part{##1}\def\@tempa{##2}}%

```

3650 \@tempb#1.\fc@end\fc@nil
3651 \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.

```

3652 \def\@tempb##1,##2\fc@nil{\def\fc@integer@part{##1}\def\@tempa{##2}}%
3653 \@tempb#1,\fc@end\fc@nil
3654 \ifx\@tempa\fc@end@

```

No comma either, so fractional part is set empty.

```

3655 \def\fc@fractional@part{}%
3656 \else

```

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

```

3657 \def\@tempb##1,\fc@end{\def\fc@fractional@part{##1}}%
3658 \expandafter\@tempb\@tempa
3659 \fi
3660 \else

```

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

```

3661 \def\@tempb##1.\fc@end{\def\fc@fractional@part{##1}}%
3662 \expandafter\@tempb\@tempa
3663 \fi
3664 }

```

\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>.

```

3665 \def\fc@number@parser#1{%

```

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

```

3666 \let\@tempa\@empty
3667 \def\@tempb##1##2\fc@nil{%
3668 \def\@tempc{##1}%
3669 \ifx\@tempc\space
3670 \else
3671 \expandafter\def\expandafter\@tempa\expandafter{\@tempa ##1}%
3672 \fi
3673 \def\@tempc{##2}%
3674 \ifx\@tempc\@empty
3675 \expandafter\@gobble
3676 \else
3677 \expandafter\@tempb
3678 \fi
3679 ##2\fc@nil
3680 }%
3681 \@tempb#1\fc@nil

```

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

```

3682 \def\@tempb##1##2\fc@nil{\def\fc@sign{##1}\def\fc@number{##2}}%

```

```

3683 \expandafter\@tempb\@tempa\fc@nil
3684 \expandafter\if\fc@sign+%
3685 \def\fc@sign@case{1}%
3686 \else
3687 \expandafter\if\fc@sign-%
3688 \def\fc@sign@case{2}%
3689 \else
3690 \def\fc@sign{}%
3691 \def\fc@sign@case{0}%
3692 \let\fc@number\@tempa
3693 \fi
3694 \fi
3695 \ifx\fc@number\@empty
3696 \PackageError{fcnumparser}{Invalid number}{Number must contain at least one non blank
3697 character after sign}%
3698 \fi

```

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

```

3699 \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 $\text{\fc@unit@weight} - 1$.

```

3700 \expandafter\fc@digit@counter\fc@unit@weight
3701 \expandafter\fc@parse@integer@digits\fc@integer@part\fc@end\fc@nil

```

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

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

```

3702 \fc@digit@counter -\fc@digit@counter
3703 \advance\fc@digit@counter by \fc@unit@weight
3704 \advance\fc@digit@counter by \fc@unit@weight
3705 \advance\fc@digit@counter by -1 %
3706 \edef\fc@max@weight{\the\fc@digit@counter}%

```

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

```

3707 {%
3708 \count0 \fc@unit@weight\relax
3709 \count1 \fc@max@weight\relax
3710 \advance\count0 by -\count1 %
3711 \advance\count0 by -1 %
3712 \def\@tempa##1{\def\@tempb{\def\@tempi{##1}}}%
3713 \expandafter\@tempa\expandafter{\the\count0}%
3714 \expandafter

```

```
3715 } \@tempb
```

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

```
3716 \expandafter\fc@digit@counter\fc@unit@weight
3717 \def\@templ{%
3718   \ifnum\fc@digit@counter>\fc@max@weight
3719     \let\next\relax
3720   \else
```

Here is the loop body:

```
3721   {%
3722     \count0 \@tempi
3723     \advance\count0 by \fc@digit@counter
3724     \expandafter\def\expandafter\@tempd\expandafter{\csname fc@digit@the\count0\endc
3725     \expandafter\def\expandafter\@tempe\expandafter{\csname fc@digit@the\fc@digit@co
3726     \def\@tempa###1###2{\def\@tempb{\let###1###2}}%
3727     \expandafter\expandafter\expandafter\@tempa\expandafter\@tempe\@tempd
3728     \expandafter
3729   }\@tempb
3730   \advance\fc@digit@counter by 1 %
3731   \fi
3732   \next
3733 }%
3734 \let\next\@templ
3735 \@templ
```

Split \fc@fractional@part into a sequence of \fc@digit@⟨*n*⟩ with ⟨*n*⟩ 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.

```
3736 \expandafter\fc@digit@counter\fc@unit@weight
3737 \expandafter\fc@parse@integer@digits\fc@fractional@part\fc@end\fc@nil
3738 \edef\fc@min@weight{\the\fc@digit@counter}%
3739 }
```

Macro \fc@parse@integer@digits is used to

```
3740 \ifcsundef\fc@parse@integer@digits\fc@{}%
3741 \PackageError{fcnumparser}{Duplicate definition}{Redefinition of
3742   macro 'fc@parse@integer@digits'}%
3743 \def\fc@parse@integer@digits#1#2\fc@nil{%
3744   \def\@tempa{#1}%
3745   \ifx\@tempa\fc@end@
3746     \def\next##1\fc@nil{}}%
3747   \else
3748     \let\next\fc@parse@integer@digits
3749     \advance\fc@digit@counter by -1
3750     \expandafter\def\csname fc@digit@the\fc@digit@counter\endcsname{#1}%
3751     \fi
3752     \next#2\fc@nil
3753 }
```

```

3754
3755
3756 \newcommand*{\fc@unit@weight}{0}
3757

```

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.

```

3758 \ifcsundef{fc@read@unit}{-}{%
3759   \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.

```

3760 \def\fc@read@unit#1#2{%
3761   \ifnum#2>\fc@max@weight
3762     #1=0\relax
3763   \else
3764     \ifnum#2<\fc@min@weight
3765       #1=0\relax
3766     \else
3767       {%
3768         \edef\@tempa{\number#2}%
3769         \count0=\@tempa
3770         \edef\@tempa{\csname fc@digit@the\count0\endcsname}%
3771         \def\@tempb##1{\def\@tempa{#1=##1\relax}}%
3772         \expandafter\@tempb\expandafter{\@tempa}%
3773         \expandafter
3774       }\@tempa
3775     \fi
3776   \fi
3777 }

```

`\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.

```

3778 \ifcsundef{fc@read@hundred}{-}{%
3779   \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

```

3780 \def\fc@read@hundred#1#2{%
3781   {%
3782     \fc@read@unit{\count0}{#2}%
3783     \def\@tempa##1{\fc@read@unit{\count1}{##1}}%
3784     \count2=#2%
3785     \advance\count2 by 1 %
3786     \expandafter\@tempa{\the\count2}%

```

```

3787 \multiply\count1 by 10 %
3788 \advance\count1 by \count0 %
3789 \def\@tempa##1{\def\@tempb{#1=##1\relax}}
3790 \expandafter\@tempa\expandafter{\the\count1}%
3791 \expandafter
3792 }\@tempb
3793 }

```

`\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.

```

3794 \ifcsundef{fc@read@thousand}{}%
3795 \PackageError{fncnumparser}{Duplicate definition}{Redefinition of macro
3796 '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

```

3797 \def\fc@read@thousand#1#2{%
3798   {%
3799     \fc@read@unit{\count0}{#2}%
3800     \def\@tempa##1{\fc@read@hundred{\count1}{##1}}%
3801     \count2=#2%
3802     \advance\count2 by 1 %
3803     \expandafter\@tempa{\the\count2}%
3804     \multiply\count1 by 10 %
3805     \advance\count1 by \count0 %
3806     \def\@tempa##1{\def\@tempb{#1=##1\relax}}
3807     \expandafter\@tempa\expandafter{\the\count1}%
3808     \expandafter
3809   }\@tempb
3810 }

```

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

```

3811 \ifcsundef{fc@read@myriad}{}%
3812 \PackageError{fncnumparser}{Duplicate definition}{Redefinition of macro
3813 '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

```

3814 \def\fc@read@myriad#1#2{%
3815   {%
3816     \fc@read@hundred{\count0}{#2}%
3817     \def\@tempa##1{\fc@read@hundred{\count1}{##1}}%
3818     \count2=#2
3819     \advance\count2 by 2
3820     \expandafter\@tempa{\the\count2}%
3821     \multiply\count1 by 100 %
3822     \advance\count1 by \count0 %

```

```

3823   \def\@tempa##1{\def\@tempb{#1=##1\relax}}%
3824   \expandafter\@tempa\expandafter{\the\count1}%
3825   \expandafter
3826   }\@tempb
3827 }

```

`\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.

```

3828 \ifcsundef{fc@check@nonzeros}{}{%
3829   \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro
3830     'fc@check@nonzeros'}}

```

Arguments as follows:

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

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

```

3831 \def\fc@check@nonzeros#1#2#3{%
3832   {%

```

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

```

3833   \edef\@@tempa{\number#1}%
3834   \edef\@tempb{\number#2}%
3835   \count0=\@@tempa
3836   \count1=\@tempb\relax

```

Then we do the real job

```

3837   \fc@@check@nonzeros@inner

```

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

```

3838   \def\@tempd##1{\def\@tempa{\def#3{##1}}}%
3839   \expandafter\@tempd\expandafter{\@tempc}%
3840   \expandafter
3841   }\@tempa
3842 }

```

`\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 seach the non-
                   zeros
         output  macro may have been redefined
\@tempc  ouput 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.
3843 \def\fc@@check@nonzeros@inner{%
3844   \ifnum\count0<\fc@min@weight
3845     \count0=\fc@min@weight\relax
3846   \fi
3847   \ifnum\count1>\fc@max@weight\relax
3848     \count1=\fc@max@weight
3849   \fi
3850   \count2\count0 %
3851   \advance\count2 by 1 %
3852   \ifnum\count0>\count1 %
3853     \PackageError{fcnumparser}{Unexpected arguments}{Number in argument 2 of macro
3854       'fc@check@nonzeros' must be at least equal to number in argument 1}%
3855   \else
3856     \fc@@check@nonzeros@inner@loopbody
3857     \ifnum\@tempc>0 %
3858       \ifnum\@tempc<9 %
3859         \ifnum\count0>\count1 %
3860         \else
3861           \let\@tempd\@tempc
3862           \fc@@check@nonzeros@inner@loopbody
3863           \ifnum\@tempc=0 %
3864             \let\@tempc\@tempd
3865           \else
3866             \def\@tempc{9}%
3867           \fi
3868         \fi
3869       \fi
3870     \fi
3871   \fi
3872 }
3873 \def\fc@@check@nonzeros@inner@loopbody{%
3874   % \@tempc <- digit of weight \count0
3875   \expandafter\let\expandafter\@tempc\csname fc@digit@the\count0\endcsname
3876   \advance\count0 by 1 %
3877   \ifnum\@tempc=0 %
3878     \ifnum\count0>\count1 %
3879     \let\next\relax

```



```

3880     \else
3881       \let\next\fc@check@nonzeros@inner@loopbody
3882     \fi
3883   \else
3884     \ifnum\count0>\count2 %
3885       \def\@tempc{9}%
3886     \fi
3887     \let\next\relax
3888   \fi
3889 \next
3890 }

```

`\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.

```

3891 \ifcsundef{fc@intpart@find@last}{}{%
3892   \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro
3893     'fc@intpart@find@last'}}

```

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

```

3894 \def\fc@intpart@find@last#1{%
3895   {%

```

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

```

3896   \count0=\fc@unit@weight\space
3897   \ifnum\count0<\fc@min@weight\space
3898     \count0=\fc@min@weight\space
3899   \fi

```

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

```

3900   \def\@templ{%
3901     \ifnum\csname fc@digit@the\count0\endcsname=0 %
3902       \advance\count0 by 1 %
3903       \ifnum\count0>\fc@max@weight\space
3904         \let\next\relax
3905       \fi
3906     \else
3907       \let\next\relax
3908     \fi
3909   \next
3910 }%
3911 \let\next\@templ
3912 \@templ

```

Now propagate result after closing bracket into counter #1.

```

3913   \toks0{#1}%
3914   \edef\@tempa{\the\toks0=\the\count0}%
3915   \expandafter
3916 }\@tempa\space

```

3917 }

`\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

```
3918 \ifcsundef{fc@get@last@word}{ }\PackageError{fcnumparser}{Duplicate definition}{Redefinitio
3919   of macro 'fc@get@last@word'}}%
3920 \def\fc@get@last@word#1#2#3{%
3921   {%
```

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`.

```
3922   \def\@tempa##1\fc@case##2\@nil\fc@end{%
3923     \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`.

```
3924     \toks1{##2\fc@case}%
3925   }%
3926   \@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`.

```
3927   \def\@tempa##1\fc@case##2\fc@end{%
3928     \toks2{##1}%
3929     \def\@tempb{##2}%
3930     \toks3{##2}%
3931   }%
```

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

```
3932   \toksdef\@tempt0 %
```

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

```
3933   \def\@templ{%
3934     \expandafter\@tempa\the\toks1 \fc@end
3935     \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.

```
3936     \let\next\relax
3937   \else
```

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

```
3938     \expandafter\expandafter\expandafter\@tempt
3939     \expandafter\expandafter\expandafter{%
3940       \expandafter\the\expandafter\@tempt
3941       \expandafter\fc@case\the\toks2}%
```

```

3942         \toks1\toks3 %
3943     \fi
3944     \next
3945 }%
3946 \let\next\@templ
3947 \@templ
3948 \edef\@tempa{\def\noexpand#2{\the\toks0}\def\noexpand#3{\the\toks2}}%
3949 \expandafter
3950 }\@tempa
3951 }

```

\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

```

3952 \ifcsundef{fc@get@last@letter}{\PackageError{fcnumparser}{Duplicate definition}{Redefinit
3953   of macro 'fc@get@last@letter'}}%
3954 \def\fc@get@last@letter#1#2#3{%
3955   {%

```

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

```

3956     \toks1{#1}%
3957     \toks0{}%
3958     \toksdef\@tempt0 %

```

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

```

3959     \def\@tempa##1##2\fc@nil{%
3960         \toks2{##1}%
3961         \toks3{##2}%
3962         \def\@tempb{##2}%
3963     }%

```

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

```

3964     \def\@templ{%
3965         \expandafter\@tempa\the\toks1 \fc@nil
3966         \ifx\@tempb\@empty

```

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

```

3967         \let\next\relax
3968     \else

```

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

```

3969         \expandafter\expandafter\expandafter\@tempt
3970         \expandafter\expandafter\expandafter{%
3971             \expandafter\the\expandafter\@tempt
3972             \the\toks2}%

```

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

```

3973         \toks1\toks3 %
3974     \fi

```

```

3975     \next
3976   }%

  Here run the loop.
3977   \let\next\@templ
3978   \next

  Now propagate the results into macros #2 and #3 after closing brace.
3979   \edef\@tempa{\def\noexpand#2{\the\toks0}\def\noexpand#3{\the\toks1}}%
3980   \expandafter
3981   }\@tempa
3982 }%

```

9.2 fcprefix.sty

Pseudo-latin prefixes.

```

3983 \NeedsTeXFormat{LaTeX2e}
3984 \ProvidesPackage{fcprefix}[2012/09/28]
3985 \RequirePackage{ifthen}
3986 \RequirePackage{keyval}
3987 \RequirePackage{fcnumparser}

```

Option ‘use duode and unde’ is to select whether 18 and suchlikes ($\langle x \rangle 8$, $\langle x \rangle 9$) writes like duodevicies, or like octodecies. For French it should be ‘below 20’.

Possible values are ‘below 20’ and ‘never’.

```

3988 \define@key{fcprefix}{use duode and unde}[below20]{%
3989   \ifthenelse{\equal{#1}{below20}}{%
3990     \def\fc@duodeandunde{2}%
3991   }{%
3992     \ifthenelse{\equal{#1}{never}}{%
3993       \def\fc@duodeandunde{0}%
3994     }{%
3995       \PackageError{fcprefix}{Unexpected option}{%
3996         Option ‘use duode and unde’ expects ‘below 20’ or ‘never’ }%
3997     }%
3998   }%
3999 }

```

Default is ‘below 20’ like in French.

```

4000 \def\fc@duodeandunde{2}

```

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

```

4001 \define@key{fcprefix}{numeral u in duo}[false]{%
4002   \ifthenelse{\equal{#1}{false}}{%
4003     \let\fc@u@in@duo\@empty
4004   }{%
4005     \ifthenelse{\equal{#1}{true}}{%
4006       \def\fc@u@in@duo{u}%
4007     }{%
4008       \PackageError{fcprefix}{Unexpected option}{%
4009         Option ‘numeral u in duo’ expects ‘true’ or ‘false’ }%

```

```

4010 }%
4011 }%
4012 }

```

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.

```

4013 \define@key{fcprefix}{e accute}[false]{%
4014   \ifthenelse{\equal{#1}{false}}{%
4015     \let\fc@prefix@eaccute\@firstofone
4016   }{%
4017     \ifthenelse{\equal{#1}{true}}{%
4018       \let\fc@prefix@eaccute\'%
4019     }{%
4020       \PackageError{fcprefix}{Unexpected option}{%
4021         Option ‘e accute’ expects ‘true’ or ‘false’ }%
4022     }%
4023   }%
4024 }

```

Default is to set accute accent like in French.

```

4025 \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 ‘millia²’

prefix for which millia squared is noted as ‘bismillia’

```

4026 \define@key{fcprefix}{power of millia}[prefix]{%
4027   \ifthenelse{\equal{#1}{prefix}}{%
4028     \let\fc@power@of@millia@init\@gobbletwo
4029     \let\fc@power@of@millia\fc@prefix@millia
4030   }{%
4031     \ifthenelse{\equal{#1}{arabic}}{%
4032       \let\fc@power@of@millia@init\@gobbletwo
4033       \let\fc@power@of@millia\fc@arabic@millia
4034     }{%
4035       \ifthenelse{\equal{#1}{recursive}}{%
4036         \let\fc@power@of@millia@init\fc@@recurse@millia@init
4037         \let\fc@power@of@millia\fc@@recurse@millia
4038       }{%
4039         \PackageError{fcprefix}{Unexpected option}{%
4040           Option ‘power of millia’ expects ‘recursive’, ‘arabic’, or ‘prefix’ }%
4041       }%
4042     }%
4043   }%
4044 }

```

Arguments as follows:

#1 output macro

#2 number with current weight *w*

```

4045 \def\fc@@recurse@millia#1#2{%

```

```

4046 \let\@temp#1%
4047 \edef#1{millia\@temp}%
4048 }

Arguments as follows — same interface as \fc@@recurse@millia:
#1 output macro
#2 number with current weight  $w$ 
4049 \def\fc@@recurse@millia@init#1#2{%
4050 {%

Save input argument current weight  $w$  into local macro \@tempb.
4051 \edef\@tempb{\number#2}%

Now main loop from 0 to  $w$ . Final value of \@tempa will be the result.
4052 \count0=0 %
4053 \let\@tempa\@empty
4054 \loop
4055 \ifnum\count0<\@tempb
4056 \advance\count0 by 1 %
4057 \expandafter\def
4058 \expandafter\@tempa\expandafter{\@tempa millia}%
4059 \repeat

Now propagate the expansion of \@tempa into #1 after closing base.
4060 \edef\@tempb{\def\noexpand#1{\@tempa}}%
4061 \expandafter
4062 }\@tempb
4063 }

Arguments as follows — same interface as \fc@@recurse@millia:
#1 output macro
#2 number with current weight  $w$ 
4064 \def\fc@@arabic@millia#1#2{%
4065 \ifnum#2=0 %
4066 \let#1\@empty
4067 \else
4068 \edef#1{millia\^{\the#2}}%
4069 \fi
4070 }

Arguments as follows — same interface as \fc@@recurse@millia:
#1 output macro
#2 number with current weight  $w$ 
4071 \def\fc@@prefix@millia#1#2{%
4072 \fc@@latin@numeral@prefix{#2}{#1}%
4073 }

Default value of option ‘power of millia’ is ‘prefix’:
4074 \let\fc@power@of@millia@init\@gobbletwo
4075 \let\fc@power@of@millia\fc@@prefix@millia

```

latin@cardinal@prefix 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. First check that macro is not yet defined.

```
4076 \ifcsundef{fc@@latin@cardinal@prefix}{}{%
4077 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro ‘fc@@latin@cardinal@p
```

Arguments as follows:

#1 input number to be formatted

#2 output macro name into which to place the formatted result

```
4078 \def\fc@@latin@cardinal@prefix#1#2{%
4079   {%
```

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

```
4080   \edef\@tempa{\number#1}%
```

Now parse number from expanded input.

```
4081   \expandafter\fc@number@parser\expandafter{\@tempa}%
4082   \count2=0 %
```

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

```
4083   \let\@tempt\@empty
4084   \def\@tempu{t}%
```

\@tempm will hold the milliaⁿ⁺³

```
4085   \let\@tempm\@empty
```

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

```
4086   \def\@templ{%
4087     \ifnum\count2>\fc@max@weight
4088       \let\next\relax
4089     \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.

```
4090     \fc@read@unit{\count3}{\count2}%
4091     \advance\count2 by 1 %
4092     \fc@read@unit{\count4}{\count2}%
4093     \advance\count2 by 1 %
4094     \fc@read@unit{\count5}{\count2}%
4095     \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.

```
4096     \edef\@tempn{%
4097       \ifnum\count3=0\else 1\fi
4098       \ifnum\count4=0\else 1\fi
4099       \ifnum\count5=0\else 1\fi
4100     }%
```

```

4101     \ifx\@tempn\@empty\else
4102         \let\@tempt\@tempu
4103         \let\@tempu\@empty
4104     \fi

```

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

```

4105     \let\@temp\@tempa
4106     \edef\@tempa{%

```

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

```

4107         \ifcase\count5 %
4108         \or cen%
4109         \or ducen%
4110         \or trecen%
4111         \or quadringen%
4112         \or quingen%
4113         \or sescen%
4114         \or septigen%
4115         \or octingen%
4116         \or nongen%
4117     \fi

```

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

```

4118         \ifnum\count4=0 %
4119         % x0(0..9)
4120         \ifnum\count2=3 %
4121         % Absolute weight zero
4122         \ifcase\count3 \@tempt
4123         \or m%
4124         \or b%
4125         \or tr%
4126         \or quadr%
4127         \or quin\@tempt
4128         \or sex\@tempt
4129         \or sep\@tempt
4130         \or oc\@tempt
4131         \or non%
4132         \fi
4133     \else

```

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

```

4134         \ifcase\count3 %
4135         \or \ifnum\count2>\fc@max@weight\else un\fi
4136         \or d\fc@u@in@duo o%
4137         \or tre%
4138         \or quattuor%
4139         \or quin%
4140         \or sex%
4141         \or septen%
4142         \or octo%

```



```

4143         \or novem%
4144         \fi
4145     \fi
4146 \else
4147     % x(10..99)
4148     \ifcase\count3 %
4149     \or un%
4150     \or d\fc@u@in@duo o%
4151     \or tre%
4152     \or quattuor%
4153     \or quin%
4154     \or sex%
4155     \or septen%
4156     \or octo%
4157     \or novem%
4158     \fi
4159     \ifcase\count4 %
4160     \or dec%
4161     \or vigin\@tempt
4162     \or trigin\@tempt
4163     \or quadragin\@tempt
4164     \or quinquagin\@tempt
4165     \or sexagin\@tempt
4166     \or septuagin\@tempt
4167     \or octogin\@tempt
4168     \or nonagin\@tempt
4169     \fi
4170 \fi

```

Insert the millia^(n÷3) only if $d_2d_1d_0 \neq 0$, i.e. if one of \count3 \count4 or \count5 is non zero.

```

4171     \@tempm

```

And append previous version of \@tempa.

```

4172     \@tempa
4173 }%

```

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

```

4174     \fc@power@of@millia\@tempm{\count2}%
4175     \fi
4176 \next
4177 }%
4178 \let\@tempa\@empty
4179 \let\@next\@templ
4180 \@templ

```

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

```

4181 \def\@tempb##1{\def\@tempa{\def#2{##1}}}%
4182 \expandafter\@tempb\expandafter{\@tempa}%

```

```

4183   \expandafter
4184 } \@tempa
4185 }

```

latin@numeral@pefix 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. First check that the macro is not yet defined.

```

4186 \ifcsundef{fc@latin@numeral@pefix}{}{%
4187   \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
4188     ‘fc@latin@numeral@pefix’}}

```

Arguments as follows:

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

```

4189 \def\fc@latin@numeral@pefix#1#2{%
4190   {%
4191     \edef\@tempa{\number#1}%
4192     \def\fc@unit@weight{0}%
4193     \expandafter\fc@number@parser\expandafter{\@tempa}%
4194     \count2=0 %

```

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

```

4195   \let\@tempm\@empty

```

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

```

4196   \def\@templ{%
4197     \ifnum\count2>\fc@max@weight
4198       \let\next\relax
4199     \else

```

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

```

4200       \fc@read@unit{\count3}{\count2}%
4201       \advance\count2 by 1 %
4202       \fc@read@unit{\count4}{\count2}%
4203       \advance\count2 by 1 %
4204       \fc@read@unit{\count5}{\count2}%
4205       \advance\count2 by 1 %

```

Check the use of duodevicies instead of octodecies.

```

4206       \let\@tempn\@secondoftwo
4207       \ifnum\count3>7 %
4208         \ifnum\count4<\fc@duodeandunde
4209           \ifnum\count4>0 %
4210             \let\@tempn\@firstoftwo
4211           \fi
4212         \fi
4213       \fi
4214       \@tempn

```

```

4215     {% use duodevicies for eighteen
4216         \advance\count4 by 1 %
4217         \let\@temps\@secondoftwo
4218     }{% do not use duodevicies for eighteen
4219         \let\@temps\@firstoftwo
4220     }%
4221     \let\@temp\@tempa
4222     \edef\@tempa{%
4223         % hundreds
4224         \ifcase\count5 %
4225         \expandafter\@gobble
4226         \or c%
4227         \or duc%
4228         \or trec%
4229         \or quadring%
4230         \or quing%
4231         \or sesc%
4232         \or septing%
4233         \or octing%
4234         \or nong%
4235         \fi
4236         {enties}%
4237         \ifnum\count4=0 %

```

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

```

4238         \ifcase\count3 %
4239         \or
4240         \ifnum\count2=3 %
4241             s\fc@prefix@eaccute emel%
4242         \else
4243             \ifnum\count2>\fc@max@weight\else un\fi
4244         \fi
4245         \or bis%
4246         \or ter%
4247         \or quater%
4248         \or quinquies%
4249         \or sexies%
4250         \or septies%
4251         \or octies%
4252         \or novies%
4253         \fi
4254     \else

```

Here $d_2 d_1 d_0$ is such that $d_1 \geq 1$.

```

4255         \ifcase\count3 %
4256         \or un%
4257         \or d\fc@u@in@duo o%
4258         \or ter%
4259         \or quater%
4260         \or quin%

```

```

4261         \or sex%
4262         \or septen%
4263         \or \@temps{octo}{duod\fc@prefix@eacute e}% x8 = two before next (x+1)0
4264         \or \@temps{novem}{und\fc@prefix@eacute e}% x9 = one before next (x+1)0
4265         \fi
4266         \ifcase\count4 %
4267         % can't get here
4268         \or d\fc@prefix@eacute ec%
4269         \or vic%
4270         \or tric%
4271         \or quadrag%
4272         \or quinquag%
4273         \or sexag%
4274         \or septuag%
4275         \or octog%
4276         \or nonag%
4277         \fi
4278         ies%
4279     \fi
4280     % Insert the millies^(n/3) only if one of \count3 \count4 \count5 is non zero
4281     \@tempm
4282     % add up previous version of \@tempa
4283     \@temppp
4284     }%

```

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

```

4285     \let\@temppp\@temppp
4286     \edef\@tempm{millies\@temppp}%
4287     \fi
4288     \next
4289 }%
4290 \let\@tempa\@empty
4291 \let\next\@templ
4292 \@templ

```

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

```

4293 \def\@tempb##1{\def\@tempa{\def#2{##1}}}%
4294 \expandafter\@tempb\expandafter{\@tempa}%
4295 \expandafter
4296 }\@tempa
4297 }

```

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 $\langle marg \rangle$ is first and $\langle oarg \rangle$ is second,
- or `\fc@call` is `\let` to be equal to `\fc@call@opt@arg@first`, and then calling convention is that the $\langle oarg \rangle$ is first and $\langle aarg \rangle$ is second,
- if $\langle oarg \rangle$ is absent, then it is by convention set empty,
- $\langle some\ macro \rangle$ is supposed to have two mandatory arguments of which $\langle oarg \rangle$ is passed to the first, and $\langle marg \rangle$ is passed to the second, and
- $\langle some\ macro \rangle$ is called within a group.

```

4298 \def\fc@call@opt@arg@second#1#2{%
4299   \def\@tempb{%
4300     \ifx[\@tempa
4301       \def\@tempc[###1]{%
4302         {#1{###1}{#2}}%
4303       }%
4304     \else
4305       \def\@tempc{#1{#2}}%
4306     \fi
4307     \@tempc
4308   }%
4309   \futurelet\@tempa
4310   \@tempb
4311 }

4312 \def\fc@call@opt@arg@first#1{%
4313   \def\@tempb{%
4314     \ifx[\@tempa
4315       \def\@tempc[###1]###2{#1{###1}{###2}}%
4316     \else
4317       \def\@tempc###1{#1{###1}}%
4318     \fi
4319     \@tempc
4320   }%
4321   \futurelet\@tempa
4322   \@tempb
4323 }
4324
4325 \let\fc@call\fc@call@opt@arg@first

```

User API.

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

- #1 local options
- #2 input number

```

4326 \newcommand*{\@latinnumeralstringnum}[2]{%
4327   \setkeys{fcprefix}{#1}%
4328   \fc@@latin@numeral@pefix{#2}\@tempa

```

```

4329 \@tempa
4330 }

Arguments as follows:
#1 local options
#2 input counter
4331 \newcommand*{\@latinnumeralstring}[2]{%
4332 \setkeys{fcprefix}{#1}%
4333 \expandafter\let\expandafter
4334 \@tempa\expandafter\csname c@#2\endcsname
4335 \expandafter\fc@latin@numeral@pefix\expandafter{\the\@tempa}\@tempa
4336 \@tempa
4337 }

4338 \newcommand*{\@latinnumeralstring}{%
4339 \fc@call\@latinnumeralstring
4340 }

4341 \newcommand*{\@latinnumeralstringnum}{%
4342 \fc@call\@latinnumeralstringnum
4343 }

```

9.3 fmtcount.sty

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

```

4344 \NeedsTeXFormat{LaTeX2e}
4345 \ProvidesPackage{fmtcount}[2015/05/05 v3.01]
4346 \RequirePackage{ifthen}

4347 \RequirePackage{xkeyval}
4348 \RequirePackage{etoolbox}
4349 \RequirePackage{fcprefix}

4350 \RequirePackage{ifxetex}

```

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

```
4351 \RequirePackage{amsgen}
```

These commands need to be defined before the configuration file is loaded.

Define the macro to format the `st`, `nd`, `rd` or `th` of an ordinal.

```
\fc@orddef@ult
```

```
4352 \providecommand*{\fc@orddef@ult}[1]{\fc@textsuperscript{#1}}
```

```
\fc@ord@multiling
```

```
4353 \providecommand*{\fc@ord@multiling}[1]{%
4354 \ifcsundef{fc@\languagenam @alias@of}{%
```

Not a supported language, just use the default setting:

```
4355 \fc@orddef@ult{#1}}{%
4356 \expandafter\let\expandafter\@tempa\csname fc@\language @alias@of\endcsname
4357 \ifcsundef{fc@ord@\@tempa}{%
```

Not language specific setting, just use the default setting:

```
4358 \fc@orddef@ult{#1}}{%
```

Language with specific setting, use that setting:

```
4359 \csname fc@ord@\@tempa\endcsname{#1}}}
```

`\padzeroes` `\padzeroes [n]`

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

```
4360 \newcount\c@padzeroesN
4361 \c@padzeroesN=1\relax
4362 \providecommand*\padzeroes}[1][17]{\c@padzeroesN=#1}
```

`\FCloadlang` `\FCloadlang{language}`

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

```
4363 \newcount\fc@tmpcatcode
4364 \def\fc@languages{}%
4365 \def\fc@mainlang{}%
4366 \newcommand*\FCloadlang}[1]{%
4367 \@FC@iflangloaded{#1}}%
4368 {%
4369 \fc@tmpcatcode=\catcode'\@ \relax
4370 \catcode '\@ 11\relax
4371 \InputIfFileExists{fc-#1.def}%
4372 {%
4373 \ifdefempty{\fc@languages}%
4374 {%
4375 \gdef\fc@languages{#1}%
4376 }%
4377 }%
4378 \gappto\fc@languages{,#1}%
4379 }%
4380 \gdef\fc@mainlang{#1}%
4381 }%
4382 }%
```

```

4383 \catcode '\@ \fc@tmpcatcode\relax
4384 }%
4385 }

```

```

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

```

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

```

4386 \newcommand{\@FC@iflangloaded}[3]{%
4387 \ifcsundef{ver@fc-#1.def}{#3}{#2}%
4388 }

```

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

```

4389 \newcommand*\ProvidesFCLanguage}[1]{%
4390 \ProvidesFile{fc-#1.def}%
4391 }

```

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

```

4392 \newif\iffmtcount@language@option
4393 \fmtcount@language@optionfalse

```

`orted@language@list` Declare list of supported languages, as a comma separated list. No space, no empty items. Each item is a language for which `fmtcount` is able to load language specific definitions. Aliases but be *after* their meaning, for instance ‘american’ being an alias of ‘USenglish’, it has to appear after it in the list. The raison d’être of this list is to commonalize iteration on languages for the two following purposes:

- loading language definition as a result of the language being used by babel/polyglossia
- loading language definition as a result of package option

These two purposes cannot be handled in the same pass, we need two different passes otherwise there would be some corner cases when a package would be required — as a result of loading language definition for one language — between a `\DeclareOption` and a `\ProcessOption` which is forbidden by $\LaTeX 2\epsilon$.

```

4394 \newcommand*\fc@supported@language@list{%
4395 english,%
4396 UKenglish,%
4397 british,%
4398 USenglish,%
4399 american,%
4400 spanish,%

```



```

4401 portuges,%
4402 french,%
4403 frenchb,%
4404 francais,%
4405 german,%
4406 germanb,%
4407 ngerman,%
4408 ngermanb,%
4409 italian}

```

iterate@on@languages

```
\fc@iterate@on@languages{<body>}
```

Now make some language iterator, note that for the following to work properly `\fc@supported@language@list` must not be empty. `<body>` is a macro that takes one argument, and `\fc@iterate@on@languages` applies it iteratively :

```

4410 \newcommand*\fc@iterate@on@languages[1]{%
4411   \ifx\fc@supported@language@list\@empty

```

That case should never happen !

```

4412   \PackageError{fmtcount}{Macro ‘\protect\fc@iterate@on@languages’ is empty}{You should :
4413     Something is broken within \texttt{fmtcount}, please report the issue on
4414     \texttt{https://github.com/search?q=fmtcount\&ref=cmdform\&type=Issues}}%
4415   \else
4416     \let\fc@iterate@on@languages@body#1
4417     \expandafter\fc@iterate@on@languages\fc@supported@language@list,\@nil,%
4418   \fi
4419 }
4420 \def\fc@iterate@on@languages#1,{%
4421   {%
4422     \def\@tempa{#1}%
4423     \ifx\@tempa\@nnil
4424       \let\@tempa\@empty
4425     \else
4426       \def\@tempa{%
4427         \fc@iterate@on@languages@body{#1}%
4428         \fc@iterate@on@languages
4429       }%
4430     \fi
4431     \expandafter
4432   }\@tempa
4433 }%

```

babelorpolyglossialdf

```
\@fc@loadifbabelorpolyglossialdf{<language>}
```

Loads `fmtcount` language file, `fc-<language>.def`, if one of the following condition is met:

- babel language definition file $\langle language \rangle$.ldf has been loaded — conditionally to compilation with latex, not xelatex.
- polyglossia language definition file gloss- $\langle language \rangle$.ldf has been loaded — conditionally to compilation with xelatex, not latex.
- $\langle language \rangle$ option has been passed to package fmtcount.

```

4434 \newcommand*{\@fc@loadifbabelorpolyglossialdf}[1]{%
4435   \ifxetex
4436     \IfFileExists{gloss-#1.ldf}{\ifcsundef{#1@loaded}}{\FCloadlang{#1}}{}}%
4437   \else
4438     \ifcsundef{ver@#1.ldf}{\FCloadlang{#1}}%
4439   \fi
4440 }

```

Load appropriate language definition files:

```
4441 \fc@iterate@on@languages\@fc@loadifbabelorpolyglossialdf
```

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

```

4442 \def\fc@iterate@on@languages@body#1{%
4443   \expandafter\def\csname fc@#1@alias@of\endcsname{#1}}
4444 \expandafter\@fc@iterate@on@languages\fc@supported@language@list,\@nil,%

```

Now define those languages that are aliases of another language. This is done with: $\@tempa\{\langle alias \rangle\}\{\langle language \rangle\}$

```

4445 \def\@tempa#1#2{%
4446   \expandafter\def\csname fc@#1@alias@of\endcsname{#2}%
4447 }%
4448 \@tempa{frenchb}{french}
4449 \@tempa{français}{french}
4450 \@tempa{germanb}{german}
4451 \@tempa{ngermanb}{german}
4452 \@tempa{ngerman}{german}
4453 \@tempa{british}{english}
4454 \@tempa{american}{USenglish}

```

Now, thanks to the aliases, we are going to define one option for each language, so that each language can have its own settings.

```

4455 \def\fc@iterate@on@languages@body#1{%
4456   \define@key{fmtcount}{#1}[]{%
4457     \@FC@iflangloaded{#1}%
4458     {%
4459       \setkeys{fc\csname fc@#1@alias@of\endcsname}{##1}%
4460     }{%
4461       \PackageError{fmtcount}%
4462         {Language ‘#1’ not defined}%
4463         {You need to load \ifxetex polyglossia\else babel\fi\space before loading fmtcount}%
4464     }%
4465   }%

```

```

4466 \ifthenelse{\equal{\csname fc@#1@alias@of\endcsname}{#1}}{%
4467   \define@key{fc\csname fc@#1@alias@of\endcsname}{fmtord}{%
4468     \ifthenelse{\equal{##1}{raise}\or\equal{##1}{level}}{%
4469       \expandafter\let\expandafter\@tempa\csname fc@set@ord@as@##1\endcsname
4470       \expandafter\@tempa\csname fc@ord@#1\endcsname
4471     }{%
4472       \ifthenelse{\equal{##1}{undefine}}{%
4473         \expandafter\let\csname fc@ord@#1\endcsname\undefined
4474       }{%
4475         \PackageError{fmtcount}%
4476         {Invalid value ‘#1’ to fmtord key}%
4477         {Option ‘fmtord’ can only take the values ‘level’, ‘raise’
4478          or ‘undefine’}%
4479       }}
4480   }%
4481 }{%

```

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

```

4482   \expandafter\let\expandafter\@tempa\csname KV@\csname fc@#1@alias@of\endcsname @fmtord\
4483   \expandafter\let\csname KV@#1@fmtord\endcsname\@tempa
4484 }%
4485 }
4486 \expandafter\@fc@iterate@on@languages\fc@supported@language@list,\@nil,%

```

fmtord Key to determine how to display the ordinal

```

4487 \def\fc@set@ord@as@level#1{%
4488   \def#1##1{##1}%
4489 }
4490 \def\fc@set@ord@as@raise#1{%
4491   \let#1\fc@textsuperscript
4492 }
4493 \define@key{fmtcount}{fmtord}{%
4494   \ifthenelse{\equal{#1}{level}
4495     \or\equal{#1}{raise}}{%
4496     {%
4497       \csname fc@set@ord@as@#1\endcsname\fc@orddef@ult
4498     }
4499   }%
4500   {%
4501     \PackageError{fmtcount}%
4502     {Invalid value ‘#1’ to fmtord key}%
4503     {Option ‘fmtord’ can only take the values ‘level’ or ‘raise’}%
4504   }%
4505 }

```

\iffmtord@abbrv Key to determine whether the ordinal superscript should be abbreviated (language dependent, currently only affects French ordinals, non-abbreviated French ordinals ending — i.e. ‘ier’ and ‘ième’ — are considered faulty.)

```

4506 \newif\iffmtord@abbrv

4507 \fmtord@abbrvtrue
4508 \define@key{fmtcount}{abbrv}[true]{%
4509   \ifthenelse{\equal{#1}{true}\or\equal{#1}{false}}{%
4510     {%
4511       \csname fmtord@abbrv#1\endcsname
4512     }%
4513     {%
4514       \PackageError{fmtcount}%
4515         {Invalid value ‘#1’ to fmtord key}%
4516         {Option ‘abbrv’ can only take the values ‘true’ or
4517           ‘false’}%
4518     }%
4519 }

```

prefix

```

4520 \define@key{fmtcount}{prefix}[scale=long]{%
4521   \RequirePackage{fmtprefix}%
4522   \fmtprefixsetoption{#1}%
4523 }

```

\fmtcountsetoptions Define command to set options.

```

4524 \def\fmtcountsetoptions{%
4525   \def\fmtcount@fmtord{}%
4526   \setkeys{fmtcount}}%

```

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

```

4527 \InputIfFileExists{fmtcount.cfg}%
4528 {%
4529   \PackageInfo{fmtcount}{Using configuration file fmtcount.cfg}%
4530 }%
4531 {%
4532 }

```

by@option@lang@list

```

4533 \newcommand*\fmtcount@loaded@by@option@lang@list{}

```

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

```

4534 \newcommand*\@fc@declare@language@option[1]{%
4535   \DeclareOption{#1}{%
4536     \ifx\fmtcount@loaded@by@option@lang@list\empty
4537       \def\fmtcount@loaded@by@option@lang@list{#1}%
4538     \else
4539       \edef\fmtcount@loaded@by@option@lang@list{\fmtcount@loaded@by@option@lang@list,#1}%
4540     \fi
4541   }%
4542 \@fc@iterate@on@languages\@fc@declare@language@option

```

level

```
4543 \DeclareOption{level}{\def\fmtcount@fmtord{level}%
4544 \def\fc@orddef@ult#1{#1}}
```

raise

```
4545 \DeclareOption{raise}{\def\fmtcount@fmtord{raise}%
4546 \def\fc@orddef@ult#1{\fc@textsuperscript{#1}}}
```

Process package options

```
4547 \ProcessOptions\relax
```

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

```
4548 \ifx\fmtcount@loaded@by@option@lang@list\@empty\else
4549 \def\fc@iterate@on@languages@body#1{%
4550 \@FC@iflangloaded{#1}{\fc@%
4551 \fmtcount@language@optiontrue
4552 \FCloadlang{#1}%
4553 }}
4554 \expandafter\@fc@iterate@on@languages\fmtcount@loaded@by@option@lang@list,\@nil,%
4555 \fi
```

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

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

```
4556 \newcount\@DT@modctr
4557 \newcommand*{\@FCmodulo}[2]{%
4558 \@DT@modctr=#1\relax
4559 \divide \@DT@modctr by #2\relax
4560 \multiply \@DT@modctr by #2\relax
4561 \advance #1 by -\@DT@modctr
4562 }
```

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

```
4563 \newcount\@ordinalctr
4564 \newcount\@orgargctr
4565 \newcount\@strctr
4566 \newcount\@tmpstrctr
```

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

```
4567 \newif\if@DT@padzeroes
4568 \newcount\@DT@loopN
4569 \newcount\@DT@X
```

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

```
4570 \newcommand*{\@binary}[1]{%
4571   \@DT@padzeroestruer
4572   \@DT@loopN=17\relax
4573   \@strctr=\@DT@loopN
4574   \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by 1}%
4575   \@strctr=65536\relax
4576   \@DT@X=#1\relax
4577   \loop
4578     \@DT@modctr=\@DT@X
4579     \divide\@DT@modctr by \@strctr
4580     \ifthenelse{\boolean{@DT@padzeroes}
4581       \and \(\@DT@modctr=0\)}
4582       \and \(\@DT@loopN>\c@padzeroesN\)}%
4583     {}%
4584     {\the\@DT@modctr}%
4585     \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
4586     \multiply\@DT@modctr by \@strctr
4587     \advance\@DT@X by -\@DT@modctr
4588     \divide\@strctr by 2\relax
4589     \advance\@DT@loopN by -1\relax
4590   \ifnum\@strctr>1
4591     \repeat
4592     \the\@DT@X
4593 }
4594
4595 \let\binarynum=\@binary
```

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

```
4596 \newcommand*{\@octal}[1]{%
4597   \ifnum#1>32768
4598     \PackageError{fmtcount}%
4599     {Value of counter too large for \protect\@octal}
4600     {Maximum value 32768}
4601   \else
4602     \@DT@padzeroestruer
4603     \@DT@loopN=6\relax
4604     \@strctr=\@DT@loopN
4605     \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by 1}%
4606     \@strctr=32768\relax
4607     \@DT@X=#1\relax
4608     \loop
4609       \@DT@modctr=\@DT@X
4610       \divide\@DT@modctr by \@strctr
4611       \ifthenelse{\boolean{@DT@padzeroes}
4612         \and \(\@DT@modctr=0\)}
4613         \and \(\@DT@loopN>\c@padzeroesN\)}%
4614       {\the\@DT@modctr}%
4615       \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
```

```

4616 \multiply\@DT@modctr by \@strctr
4617 \advance\@DT@X by -\@DT@modctr
4618 \divide\@strctr by 8\relax
4619 \advance\@DT@loopN by -1\relax
4620 \ifnum\@strctr>1
4621 \repeat
4622 \the\@DT@X
4623 \fi
4624 }
4625 \let\octalnum=\@octal

```

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

```

4626 \newcommand*\@hexadecimal}[1]{%
4627 \ifcase#1\or1\or2\or3\or4\or5\or
4628 6\or7\or8\or9\or a\or b\or c\or d\or e\or f\fi
4629 }

```

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

```

4630 \newcommand*\hexadecimal}[1]{%
4631 \@DT@padzeroestrue
4632 \@DT@loopN=5\relax
4633 \@strctr=\@DT@loopN
4634 \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by 1}%
4635 \@strctr=65536\relax
4636 \@DT@X=#1\relax
4637 \loop
4638 \@DT@modctr=\@DT@X
4639 \divide\@DT@modctr by \@strctr
4640 \ifthenelse{\boolean{\@DT@padzeroes}
4641 \and \!(\@DT@modctr=0\)}
4642 \and \!(\@DT@loopN>\c@padzeroesN\)}
4643 {\@hexadecimal\@DT@modctr}%
4644 \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
4645 \multiply\@DT@modctr by \@strctr
4646 \advance\@DT@X by -\@DT@modctr
4647 \divide\@strctr by 16\relax
4648 \advance\@DT@loopN by -1\relax
4649 \ifnum\@strctr>1
4650 \repeat
4651 \@hexadecimal\@DT@X
4652 }
4653 \let\hexadecimalnum=\hexadecimal

```

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

```

4654 \newcommand*\@Hexadecimal}[1]{%
4655 \ifcase#1\or1\or2\or3\or4\or5\or6\or
4656 7\or8\or9\or A\or B\or C\or D\or E\or F\fi
4657 }

```

`\Hexadecimalnum` Uppercase hexadecimal

```
4658 \newcommand*{\@Hexadecimal}[1]{%
4659   \@DT@padzeroestrue
4660   \@DT@loopN=5\relax
4661   \@strctr=\@DT@loopN
4662   \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by 1}%
4663   \@strctr=65536\relax
4664   \@DT@X=#1\relax
4665   \loop
4666     \@DT@modctr=\@DT@X
4667     \divide\@DT@modctr by \@strctr
4668     \ifthenelse{\boolean{@DT@padzeroes}
4669       \and \(\@DT@modctr=0\}
4670       \and \(\@DT@loopN>\c@padzeroesN\)}%
4671     {\@Hexadecimal\@DT@modctr}%
4672     \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
4673     \multiply\@DT@modctr by \@strctr
4674     \advance\@DT@X by -\@DT@modctr
4675     \divide\@strctr by 16\relax
4676     \advance\@DT@loopN by -1\relax
4677   \ifnum\@strctr>1
4678     \repeat
4679   \@Hexadecimal\@DT@X
4680 }
4681
4682 \let\Hexadecimalnum=\@Hexadecimal
```

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

```
4683 \newcommand*{\@aaalph}[1]{%
4684   \@DT@loopN=#1\relax
4685   \advance\@DT@loopN by -1\relax
4686   \divide\@DT@loopN by 26\relax
4687   \@DT@modctr=\@DT@loopN
4688   \multiply\@DT@modctr by 26\relax
4689   \@DT@X=#1\relax
4690   \advance\@DT@X by -1\relax
4691   \advance\@DT@X by -\@DT@modctr
4692   \advance\@DT@loopN by 1\relax
4693   \advance\@DT@X by 1\relax
4694   \loop
4695     \@alph\@DT@X
4696     \advance\@DT@loopN by -1\relax
4697   \ifnum\@DT@loopN>0
4698     \repeat
4699 }
4700
4701 \let\aaalphnum=\@aaalph
```

`\AAAAlphnum` Uppercase alphabetical representation (a ... z aa ... zz)


```

4702 \newcommand*{\@AAAlph}[1]{%
4703   \@DT@loopN=#1\relax
4704   \advance\@DT@loopN by -1\relax
4705   \divide\@DT@loopN by 26\relax
4706   \@DT@modctr=\@DT@loopN
4707   \multiply\@DT@modctr by 26\relax
4708   \@DT@X=#1\relax
4709   \advance\@DT@X by -1\relax
4710   \advance\@DT@X by -\@DT@modctr
4711   \advance\@DT@loopN by 1\relax
4712   \advance\@DT@X by 1\relax
4713   \loop
4714     \@Alph\@DT@X
4715     \advance\@DT@loopN by -1\relax
4716   \ifnum\@DT@loopN>0
4717     \repeat
4718 }
4719
4720 \let\AAAlphnum=\@AAAlph

```

\abalphnum Lowercase alphabetical representation

```

4721 \newcommand*{\@abalph}[1]{%
4722   \ifnum#1>17576\relax
4723     \PackageError{fmtcount}%
4724     {Value of counter too large for \protect\@abalph}%
4725     {Maximum value 17576}%
4726   \else
4727     \@DT@padzeroestrue
4728     \@strctr=17576\relax
4729     \@DT@X=#1\relax
4730     \advance\@DT@X by -1\relax
4731     \loop
4732       \@DT@modctr=\@DT@X
4733       \divide\@DT@modctr by \@strctr
4734       \ifthenelse{\boolean{\@DT@padzeroes}
4735         \and \(\@DT@modctr=1\)}%
4736         {\@alph\@DT@modctr}%
4737       \ifnum\@DT@modctr=1\else\@DT@padzeroesfalse\fi
4738       \multiply\@DT@modctr by \@strctr
4739       \advance\@DT@X by -\@DT@modctr
4740       \divide\@strctr by 26\relax
4741     \ifnum\@strctr>1
4742       \repeat
4743     \advance\@DT@X by 1\relax
4744     \@alph\@DT@X
4745   \fi
4746 }
4747
4748 \let\abalphnum=\@abalph

```

`\ABAlphnum` Uppercase alphabetical representation

```
4749 \newcommand*{\@ABAlph}[1]{%
4750   \ifnum#1>17576\relax
4751     \PackageError{fmtcount}%
4752     {Value of counter too large for \protect\@ABAlph}%
4753     {Maximum value 17576}%
4754   \else
4755     \@DT@padzeroestrue
4756     \@strctr=17576\relax
4757     \@DT@X=#1\relax
4758     \advance\@DT@X by -1\relax
4759     \loop
4760       \@DT@modctr=\@DT@X
4761       \divide\@DT@modctr by \@strctr
4762       \ifthenelse{\boolean{\@DT@padzeroes}\and
4763         \(\@DT@modctr=1\)}{\@Alph\@DT@modctr}%
4764       \ifnum\@DT@modctr=1\else\@DT@padzeroesfalse\fi
4765       \multiply\@DT@modctr by \@strctr
4766       \advance\@DT@X by -\@DT@modctr
4767       \divide\@strctr by 26\relax
4768     \ifnum\@strctr>1
4769     \repeat
4770     \advance\@DT@X by 1\relax
4771     \@Alph\@DT@X
4772   \fi
4773 }
4774
4775 \let\ABAlphnum=\@ABAlph
```

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

```
4776 \def\@fmtc@count#1#2\relax{%
4777   \if\relax#1%
4778   \else
4779     \advance\@strctr by 1\relax
4780     \@fmtc@count#2\relax
4781   \fi
4782 }
```

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

```
4783 \newcommand{\@decimal}[1]{%
4784   \@strctr=0\relax
4785   \expandafter\@fmtc@count\number#1\relax
4786   \@DT@loopN=\c@padzeroesN
4787   \advance\@DT@loopN by -\@strctr
4788   \ifnum\@DT@loopN>0\relax
4789     \@strctr=0\relax
4790     \whiledo{\@strctr < \@DT@loopN}{\advance\@strctr by 1\relax}%
4791   \fi
```

```

4792 \number#1\relax
4793 }
4794
4795 \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.

```

4796 \newcommand{\FCordinal}[1]{%
4797   \expandafter\protect\expandafter\ordinalnum{%
4798     \expandafter\the\csname c@#1\endcsname}%
4799 }

```

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

```

4800 \ifcsundef{ordinal}
4801 {\let\ordinal\FCordinal}%
4802 {%
4803   \PackageWarning{fmtcount}%
4804   {\protect\ordinal \space already defined use
4805    \protect\FCordinal \space instead.}
4806 }

```

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

```

4807 \newcommand*{\ordinalnum}[1]{%
4808   \new@ifnextchar[%
4809     {\@ordinalnum{#1}}%
4810     {\@ordinalnum{#1}[m]}%
4811 }

```

\@ordinalnum Display ordinal according to gender (neuter added in v1.1, \xspace added in v1.2, and removed in v1.3⁷):

```

4812 \def\@ordinalnum#1[#2]{%
4813   {%

```

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

```

4814 \ifthenelse{\equal{#2}{f}}%
4815 {%
4816 \protect\@ordinalF{#1}{\@fc@ordstr}%
4817 }%
4818 {%
4819 \ifthenelse{\equal{#2}{n}}%
4820 {%
4821 \protect\@ordinalN{#1}{\@fc@ordstr}%
4822 }%
4823 {%
4824 \ifthenelse{\equal{#2}{m}}%
4825 {}%
4826 {%
4827 \PackageError{fmtcount}%
4828 {Invalid gender option '#2'}%
4829 {Available options are m, f or n}%
4830 }%
4831 \protect\@ordinalM{#1}{\@fc@ordstr}%
4832 }%
4833 }%
4834 \@fc@ordstr
4835 }%
4836 }

```

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

```

4837 \newcommand*\storeordinal}[2]{%
4838 \expandafter\protect\expandafter\storeordinalnum{#1}{%
4839 \expandafter\the\csname c@#2\endcsname}%
4840 }

```

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

```

4841 \newcommand*\storeordinalnum}[2]{%
4842 \@ifnextchar[%
4843 {\storeordinalnum{#1}{#2}}%
4844 {\storeordinalnum{#1}{#2}[m]}%
4845 }

```

`\@storeordinalnum` Store ordinal according to gender:

```

4846 \def\@storeordinalnum#1#2[#3]{%
4847 \ifthenelse{\equal{#3}{f}}%
4848 {%
4849 \protect\@ordinalF{#2}{\@fc@ord}
4850 }%
4851 {%
4852 \ifthenelse{\equal{#3}{n}}%
4853 {%
4854 \protect\@ordinalN{#2}{\@fc@ord}%

```

```

4855 }%
4856 {%
4857 \ifthenelse{\equal{#3}{m}}%
4858 {}%
4859 {%
4860 \PackageError{fmtcount}%
4861 {Invalid gender option ‘#3’}%
4862 {Available options are m or f}%
4863 }%
4864 \protect\ordinalM{#2}{\@fc@ord}%
4865 }%
4866 }%
4867 \expandafter\let\csname @fcs@#1\endcsname\@fc@ord
4868 }

```

`\FMCuse` Get stored information:

```

4869 \newcommand*\FMCuse[1]{\csname @fcs@#1\endcsname}

```

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

```

4870 \newcommand*\ordinalstring[1]{%
4871 \expandafter\protect\expandafter\ordinalstringnum{%
4872 \expandafter\the\csname c@#1\endcsname}%
4873 }

```

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

```

4874 \newcommand\ordinalstringnum[1]{%
4875 \new@ifnextchar[%
4876 {\@ordinal@string{#1}}%
4877 {\@ordinal@string{#1}[m]}%
4878 }

```

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

```

4879 \def\@ordinal@string#1[#2]{%
4880 {%
4881 \ifthenelse{\equal{#2}{f}}%
4882 {%
4883 \protect\ordinalstringF{#1}{\@fc@ordstr}%
4884 }%
4885 {%
4886 \ifthenelse{\equal{#2}{n}}%
4887 {%
4888 \protect\ordinalstringN{#1}{\@fc@ordstr}%
4889 }%
4890 {%
4891 \ifthenelse{\equal{#2}{m}}%
4892 {}%
4893 {%
4894 \PackageError{fmtcount}%
4895 {Invalid gender option ‘#2’ to \protect\ordinalstring}%

```

```

4896         {Available options are m, f or n}%
4897     }%
4898     \protect\@ordinalstringM{#1}{\@fc@ordstr}%
4899 }%
4900 }%
4901     \@fc@ordstr
4902 }%
4903 }

```

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

```

4904 \newcommand*\storeordinalstring}[2]{%
4905   \expandafter\protect\expandafter\storeordinalstringnum{#1}{%
4906     \expandafter\the\csname c@#2\endcsname}%
4907 }

```

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

```

4908 \newcommand*\storeordinalstringnum}[2]{%
4909   \@ifnextchar[%
4910     {\@store@ordinal@string{#1}{#2}}%
4911     {\@store@ordinal@string{#1}{#2}[m]}%
4912 }

```

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

```

4913 \def\@store@ordinal@string#1#2[#3]{%
4914   \ifthenelse{\equal{#3}{f}}%
4915   {%
4916     \protect\@ordinalstringF{#2}{\@fc@ordstr}%
4917   }%
4918   {%
4919     \ifthenelse{\equal{#3}{n}}%
4920     {%
4921       \protect\@ordinalstringN{#2}{\@fc@ordstr}%
4922     }%
4923     {%
4924       \ifthenelse{\equal{#3}{m}}%
4925       {}%
4926       {%
4927         \PackageError{fmtcount}%
4928           {Invalid gender option ‘#3’ to \protect\ordinalstring}%
4929           {Available options are m, f or n}%
4930       }%
4931       \protect\@ordinalstringM{#2}{\@fc@ordstr}%
4932     }%
4933   }%
4934   \expandafter\let\csname @fcs@#1\endcsname\@fc@ordstr
4935 }

```

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

```
4936 \newcommand*\Ordinalstring[1]{%
4937   \expandafter\protect\expandafter\Ordinalstringnum{%
4938     \expandafter\the\csname c@#1\endcsname}%
4939 }
```

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

```
4940 \newcommand*\Ordinalstringnum[1]{%
4941   \new@ifnextchar[%
4942     {\@Ordinal@string{#1}}%
4943     {\@Ordinal@string{#1}[m]}%
4944 }
```

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

```
4945 \def\@Ordinal@string#1[#2]{%
4946   {%
4947     \ifthenelse{\equal{#2}{f}}%
4948     {%
4949       \protect\@OrdinalstringF{#1}{\@fc@ordstr}%
4950     }%
4951     {%
4952       \ifthenelse{\equal{#2}{n}}%
4953       {%
4954         \protect\@OrdinalstringN{#1}{\@fc@ordstr}%
4955       }%
4956       {%
4957         \ifthenelse{\equal{#2}{m}}%
4958         {}%
4959         {%
4960           \PackageError{fmtcount}%
4961             {Invalid gender option ‘#2’}%
4962             {Available options are m, f or n}%
4963           }%
4964           \protect\@OrdinalstringM{#1}{\@fc@ordstr}%
4965         }%
4966       }%
4967     \@fc@ordstr
4968   }%
4969 }
```

`\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.

```
4970 \newcommand*\storeOrdinalstring[2]{%
4971   \expandafter\protect\expandafter\storeOrdinalstringnum{#1}{%
4972     \expandafter\the\csname c@#2\endcsname}%
4973 }
```

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

```
4974 \newcommand*\storeOrdinalstringnum[2]{%
4975   \@ifnextchar[%
4976     {\@store@Ordinal@string{#1}{#2}}%
4977     {\@store@Ordinal@string{#1}{#2}[m]}%
4978 }
```

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

```
4979 \def\@store@Ordinal@string#1#2[#3]{%
4980   \ifthenelse{\equal{#3}{f}}%
4981     {%
4982       \protect\@OrdinalstringF{#2}{\@fc@ordstr}%
4983     }%
4984     {%
4985       \ifthenelse{\equal{#3}{n}}%
4986         {%
4987           \protect\@OrdinalstringN{#2}{\@fc@ordstr}%
4988         }%
4989         {%
4990           \ifthenelse{\equal{#3}{m}}%
4991             {}%
4992             {%
4993               \PackageError{fmtcount}%
4994                 {Invalid gender option ‘#3’}%
4995                 {Available options are m or f}%
4996             }%
4997           \protect\@OrdinalstringM{#2}{\@fc@ordstr}%
4998         }%
4999     }%
5000 \expandafter\let\csname @fcs@#1\endcsname\@fc@ordstr
5001 }
```

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

```
5002 \newcommand*\storeORDINALstring[2]{%
5003   \expandafter\protect\expandafter\storeORDINALstringnum{#1}{%
5004     \expandafter\the\csname c@#2\endcsname}%
5005 }
```

oreORDINALstringnum As above, but the second argument is a count register or a number.

```
5006 \newcommand*\storeORDINALstringnum[2]{%
5007   \@ifnextchar[%
5008     {\@store@ORDINAL@string{#1}{#2}}%
5009     {\@store@ORDINAL@string{#1}{#2}[m]}%
5010 }
```


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

```
5011 \def\@store@ORDINAL@string#1#2[#3]{%
5012   \ifthenelse{\equal{#3}{f}}{%
5013     {%
5014       \protect\@ordinalstringF{#2}{\@fc@ordstr}%
5015     }%
5016     {%
5017       \ifthenelse{\equal{#3}{n}}{%
5018         {%
5019           \protect\@ordinalstringN{#2}{\@fc@ordstr}%
5020         }%
5021         {%
5022           \ifthenelse{\equal{#3}{m}}{%
5023             {}%
5024             {%
5025               \PackageError{fmtcount}%
5026                 {Invalid gender option ‘#3’}%
5027                 {Available options are m or f}%
5028             }%
5029           \protect\@ordinalstringM{#2}{\@fc@ordstr}%
5030         }%
5031       }%
5032     \expandafter\protected@edef\csname @fcs@#1\endcsname{%
5033       \noexpand\MakeUppercase{\@fc@ordstr}%
5034     }%
5035 }
```

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

```
5036 \newcommand*\ORDINALstring}[1]{%
5037   \expandafter\protect\expandafter\ORDINALstringnum{%
5038     \expandafter\the\csname c@#1\endcsname
5039   }%
5040 }
```

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

```
5041 \newcommand*\ORDINALstringnum}[1]{%
5042   \new@ifnextchar[%
5043     {\@ORDINAL@string{#1}}%
5044     {\@ORDINAL@string{#1}[m]}%
5045 }
```

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

```
5046 \def\@ORDINAL@string#1[#2]{%
5047   {%
5048     \ifthenelse{\equal{#2}{f}}{%
5049       {%
5050         \protect\@ordinalstringF{#1}{\@fc@ordstr}%
5051       }%
5052     }%
5053     {%
5054       \ifthenelse{\equal{#2}{n}}{%
5055         {%
5056           \protect\@ordinalstringN{#1}{\@fc@ordstr}%
5057         }%
5058         {%
5059           \ifthenelse{\equal{#2}{m}}{%
5060             {}%
5061             {%
5062               \PackageError{fmtcount}%
5063                 {Invalid gender option ‘#2’}%
5064                 {Available options are m or f}%
5065             }%
5066           \protect\@ordinalstringM{#1}{\@fc@ordstr}%
5067         }%
5068       }%
5069     }%
5070   \expandafter\protected@edef\csname @fcs@#1\endcsname{%
5071     \noexpand\MakeUppercase{\@fc@ordstr}%
5072   }%
5073 }
```

```

5051 }%
5052 {%
5053   \ifthenelse{\equal{#2}{n}}%
5054   {%
5055     \protect\ordinalstringN{#1}{\@fc@ordstr}%
5056   }%
5057   {%
5058     \ifthenelse{\equal{#2}{m}}%
5059     {}%
5060     {%
5061       \PackageError{fmtcount}%
5062       {Invalid gender option ‘#2’}%
5063       {Available options are m, f or n}%
5064     }%
5065     \protect\ordinalstringM{#1}{\@fc@ordstr}%
5066   }%
5067 }%
5068 \MakeUppercase{\@fc@ordstr}%
5069 }%
5070 }

```

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

```

5071 \newcommand*\storenumberstring[2]{%
5072   \expandafter\protect\expandafter\storenumberstringnum{#1}{%
5073     \expandafter\the\csname c@#2\endcsname}%
5074 }

```

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

```

5075 \newcommand\storenumberstringnum[2]{%
5076   \@ifnextchar[%
5077   {\@store@number@string{#1}{#2}}%
5078   {\@store@number@string{#1}{#2}[m]}%
5079 }

```

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

```

5080 \def\@store@number@string#1#2[#3]{%
5081   \ifthenelse{\equal{#3}{f}}%
5082   {%
5083     \protect\@numberstringF{#2}{\@fc@numstr}%
5084   }%
5085   {%
5086     \ifthenelse{\equal{#3}{n}}%
5087     {%
5088       \protect\@numberstringN{#2}{\@fc@numstr}%
5089     }%
5090     {%
5091       \ifthenelse{\equal{#3}{m}}%
5092       {}%

```

```

5093     {%
5094     \PackageError{fmtcount}
5095     {Invalid gender option ‘#3’}%
5096     {Available options are m, f or n}%
5097     }%
5098     \protect\@numberstringM{#2}{\@fc@numstr}%
5099     }%
5100 }%
5101 \expandafter\let\csname @fcs@#1\endcsname\@fc@numstr
5102 }

```

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

```

5103 \newcommand*{\numberstring}[1]{%
5104 \expandafter\protect\expandafter\numberstringnum{%
5105 \expandafter\the\csname c@#1\endcsname}%
5106 }

```

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

```

5107 \newcommand*{\numberstringnum}[1]{%
5108 \new@ifnextchar[%
5109 {\@number@string{#1}}%
5110 {\@number@string{#1}[m]}%
5111 }

```

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

```

5112 \def\@number@string#1[#2]{%
5113   {%
5114     \ifthenelse{\equal{#2}{f}}%
5115     {%
5116       \protect\@numberstringF{#1}{\@fc@numstr}%
5117     }%
5118     {%
5119       \ifthenelse{\equal{#2}{n}}%
5120       {%
5121         \protect\@numberstringN{#1}{\@fc@numstr}%
5122       }%
5123       {%
5124         \ifthenelse{\equal{#2}{m}}%
5125         {}%
5126         {%
5127           \PackageError{fmtcount}%
5128           {Invalid gender option ‘#2’}%
5129           {Available options are m, f or n}%
5130         }%
5131         \protect\@numberstringM{#1}{\@fc@numstr}%
5132       }%
5133     }%
5134     \@fc@numstr
5135   }%

```

5136 }

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

```
5137 \newcommand*\storeNumberstring}[2]{%
5138   \expandafter\protect\expandafter\storeNumberstringnum{#1}{%
5139     \expandafter\the\csname c@#2\endcsname}%
5140 }
```

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

```
5141 \newcommand*\storeNumberstringnum}[2]{%
5142   \@ifnextchar[%
5143     {\@store@Number@string{#1}{#2}}%
5144     {\@store@Number@string{#1}{#2}[m]}%
5145 }
```

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

```
5146 \def\@store@Number@string#1#2[#3]{%
5147   \ifthenelse{\equal{#3}{f}}{%
5148     {%
5149       \protect\@NumberstringF{#2}{\@fc@numstr}%
5150     }%
5151     {%
5152       \ifthenelse{\equal{#3}{n}}{%
5153         {%
5154           \protect\@NumberstringN{#2}{\@fc@numstr}%
5155         }%
5156         {%
5157           \ifthenelse{\equal{#3}{m}}{%
5158             {}%
5159             {%
5160               \PackageError{fmtcount}%
5161                 {Invalid gender option ‘#3’}%
5162                 {Available options are m, f or n}%
5163             }%
5164           \protect\@NumberstringM{#2}{\@fc@numstr}%
5165         }%
5166       }%
5167     \expandafter\let\csname @fcs@#1\endcsname\@fc@numstr
5168 }
```

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

```
5169 \newcommand*\Numberstring}[1]{%
5170   \expandafter\protect\expandafter\Numberstringnum{%
5171     \expandafter\the\csname c@#1\endcsname}%
5172 }
```

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

```

5173 \newcommand*\Numberstringnum}[1]{%
5174   \new@ifnextchar[%
5175     {\@Number@string{#1}}}%
5176     {\@Number@string{#1}[m]}%
5177 }

```

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

```

5178 \def\@Number@string#1[#2]{%
5179   {%
5180     \ifthenelse{\equal{#2}{f}}%
5181     {%
5182       \protect\@NumberstringF{#1}{\@fc@numstr}%
5183     }%
5184     {%
5185       \ifthenelse{\equal{#2}{n}}%
5186       {%
5187         \protect\@NumberstringN{#1}{\@fc@numstr}%
5188       }%
5189       {%
5190         \ifthenelse{\equal{#2}{m}}%
5191         {}%
5192         {%
5193           \PackageError{fmtcount}%
5194             {Invalid gender option ‘#2’}%
5195             {Available options are m, f or n}%
5196         }%
5197         \protect\@NumberstringM{#1}{\@fc@numstr}%
5198       }%
5199     }%
5200   \@fc@numstr
5201 }%
5202 }

```

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

```

5203 \newcommand{\storeNUMBERstring}[2]{%
5204   \expandafter\protect\expandafter\storeNUMBERstringnum{#1}{%
5205     \expandafter\the\csname c@#2\endcsname}%
5206 }

```

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

```

5207 \newcommand{\storeNUMBERstringnum}[2]{%
5208   \@ifnextchar[%
5209     {\@store@NUMBER@string{#1}{#2}}%
5210     {\@store@NUMBER@string{#1}{#2}[m]}%
5211 }

```

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

```

5212 \def\@store@NUMBER@string#1#2[#3]{%

```

```

5213 \ifthenelse{\equal{#3}{f}}%
5214 {%
5215   \protect\@numberstringF{#2}{\@fc@numstr}%
5216 }%
5217 {%
5218   \ifthenelse{\equal{#3}{n}}%
5219   {%
5220     \protect\@numberstringN{#2}{\@fc@numstr}%
5221   }%
5222   {%
5223     \ifthenelse{\equal{#3}{m}}%
5224     {}%
5225     {%
5226       \PackageError{fmtcount}%
5227       {Invalid gender option ‘#3’}%
5228       {Available options are m or f}%
5229     }%
5230     \protect\@numberstringM{#2}{\@fc@numstr}%
5231   }%
5232 }%
5233 \expandafter\edef\csname @fcs@#1\endcsname{%
5234   \noexpand\MakeUppercase{\@fc@numstr}%
5235 }%
5236 }

```

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

```

5237 \newcommand*{\NUMBERstring}[1]{%
5238   \expandafter\protect\expandafter\NUMBERstringnum{%
5239     \expandafter\the\csname c@#1\endcsname}%
5240 }

```

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

```

5241 \newcommand*{\NUMBERstringnum}[1]{%
5242   \new@ifnextchar[%
5243     {\@NUMBER@string{#1}}%
5244     {\@NUMBER@string{#1}[m]}%
5245 }

```

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

```

5246 \def\@NUMBER@string#1[#2]{%
5247   {%
5248     \ifthenelse{\equal{#2}{f}}%
5249     {%
5250       \protect\@numberstringF{#1}{\@fc@numstr}%
5251     }%
5252     {%
5253       \ifthenelse{\equal{#2}{n}}%
5254       {%

```

```

5255         \protect\@numberstringN{#1}{\@fc@numstr}%
5256     }%
5257     {%
5258         \ifthenelse{\equal{#2}{m}}%
5259         {}%
5260         {%
5261             \PackageError{fmtcount}%
5262             {Invalid gender option ‘#2’}%
5263             {Available options are m, f or n}%
5264         }%
5265         \protect\@numberstringM{#1}{\@fc@numstr}%
5266     }%
5267 }%
5268 \MakeUppercase{\@fc@numstr}%
5269 }%
5270 }

```

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

```

5271 \providecommand*\binary}[1]{%
5272     \expandafter\protect\expandafter\@binary{%
5273         \expandafter\the\csname c@#1\endcsname}%
5274 }

```

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

```

5275 \providecommand*\aaalph}[1]{%
5276     \expandafter\protect\expandafter\@aaalph{%
5277         \expandafter\the\csname c@#1\endcsname}%
5278 }

```

`\AAAAlph` As before, but upper case.

```

5279 \providecommand*\AAAAlph}[1]{%
5280     \expandafter\protect\expandafter\@AAAAlph{%
5281         \expandafter\the\csname c@#1\endcsname}%
5282 }

```

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

```

5283 \providecommand*\abalph}[1]{%
5284     \expandafter\protect\expandafter\@abalph{%
5285         \expandafter\the\csname c@#1\endcsname}%
5286 }

```

`\ABAlph` As above, but upper case.

```

5287 \providecommand*\ABAlph}[1]{%
5288     \expandafter\protect\expandafter\@ABAlph{%
5289         \expandafter\the\csname c@#1\endcsname}%
5290 }

```

`\hexadecimal` Hexadecimal:

```

5291 \providecommand*\hexadecimal}[1]{%
5292   \expandafter\protect\expandafter\@hexadecimal{%
5293     \expandafter\the\csname c@#1\endcsname}%
5294 }

```

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

```

5295 \providecommand*\Hexadecimal}[1]{%
5296   \expandafter\protect\expandafter\@Hexadecimal{%
5297     \expandafter\the\csname c@#1\endcsname}%
5298 }

```

`\octal` Octal:

```

5299 \providecommand*\octal}[1]{%
5300   \expandafter\protect\expandafter\@octal{%
5301     \expandafter\the\csname c@#1\endcsname}%
5302 }

```

`\decimal` Decimal:

```

5303 \providecommand*\decimal}[1]{%
5304   \expandafter\protect\expandafter\@decimal{%
5305     \expandafter\the\csname c@#1\endcsname}%
5306 }

```

9.4 Multilanguage 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.

```

5307 \def\@setdef@ultfmtcount{%
5308   \ifcsundef\@ordinalMenglish{\FCloadlang{english}}{ }%
5309   \def\@ordinalstringM{\@ordinalstringMenglish}%
5310   \let\@ordinalstringF=\@ordinalstringMenglish
5311   \let\@ordinalstringN=\@ordinalstringMenglish
5312   \def\@OrdinalstringM{\@OrdinalstringMenglish}%
5313   \let\@OrdinalstringF=\@OrdinalstringMenglish
5314   \let\@OrdinalstringN=\@OrdinalstringMenglish
5315   \def\@numberstringM{\@numberstringMenglish}%
5316   \let\@numberstringF=\@numberstringMenglish
5317   \let\@numberstringN=\@numberstringMenglish
5318   \def\@NumberstringM{\@NumberstringMenglish}%
5319   \let\@NumberstringF=\@NumberstringMenglish
5320   \let\@NumberstringN=\@NumberstringMenglish
5321   \def\@ordinalM{\@ordinalMenglish}%
5322   \let\@ordinalF=\@ordinalM
5323   \let\@ordinalN=\@ordinalM
5324   \let\fmtord\fcorddef@ult
5325 }

```



```

\fc@multiling \fc@multiling{<name>}{<gender>}
5326 \newcommand*{\fc@multiling}[2]{%
5327 \ifcsundef{@#1#2\languagename}%
5328 {% try loading it
5329 \FCloadlang{\languagename}%
5330 }%
5331 {%
5332 }%
5333 \ifcsundef{@#1#2\languagename}%
5334 {%
5335 \PackageWarning{fmtcount}%
5336 {No support for \expandafter\protect\csname #1\endcsname\space for
5337 language '\languagename'}%
5338 \ifthenelse{\equal{\languagename}{\fc@mainlang}}{%
5339 {%
5340 \FCloadlang{english}%
5341 }%
5342 {%
5343 }%
5344 \ifcsdef{@#1#2\fc@mainlang}%
5345 {%
5346 \csuse{@#1#2\fc@mainlang}%
5347 }%
5348 {%
5349 \PackageWarningNoLine{fmtcount}%
5350 {No languages loaded at all! Loading english definitions}%
5351 \FCloadlang{english}%
5352 \def\fc@mainlang{english}%
5353 \csuse{@#1#2english}%
5354 }%
5355 }%
5356 {%
5357 \csuse{@#1#2\languagename}%
5358 }%
5359 }

```

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

```

5360 \def\@set@multiling@fmtcount{%
    The masculine version of \numberstring:
5361 \def\@numberstringM{%
5362 \fc@multiling{numberstring}{M}%
5363 }%
    The feminine version of \numberstring:
5364 \def\@numberstringF{%
5365 \fc@multiling{numberstring}{F}%
5366 }%

```

The neuter version of \numberstring:

5367 \def\@numberstringN{%
5368 \fc@multiling{numberstring}{N}%
5369 }%

The masculine version of \Numberstring:

5370 \def\@NumberstringM{%
5371 \fc@multiling{Numberstring}{M}%
5372 }%

The feminine version of \Numberstring:

5373 \def\@NumberstringF{%
5374 \fc@multiling{Numberstring}{F}%
5375 }%

The neuter version of \Numberstring:

5376 \def\@NumberstringN{%
5377 \fc@multiling{Numberstring}{N}%
5378 }%

The masculine version of \ordinal:

5379 \def\@ordinalM{%
5380 \fc@multiling{ordinal}{M}%
5381 }%

The feminine version of \ordinal:

5382 \def\@ordinalF{%
5383 \fc@multiling{ordinal}{F}%
5384 }%

The neuter version of \ordinal:

5385 \def\@ordinalN{%
5386 \fc@multiling{ordinal}{N}%
5387 }%

The masculine version of \ordinalstring:

5388 \def\@ordinalstringM{%
5389 \fc@multiling{ordinalstring}{M}%
5390 }%

The feminine version of \ordinalstring:

5391 \def\@ordinalstringF{%
5392 \fc@multiling{ordinalstring}{F}%
5393 }%

The neuter version of \ordinalstring:

5394 \def\@ordinalstringN{%
5395 \fc@multiling{ordinalstring}{N}%
5396 }%

The masculine version of \Ordinalstring:

5397 \def\@OrdinalstringM{%
5398 \fc@multiling{Ordinalstring}{M}%
5399 }%

The feminine version of \Ordinalstring:

```
5400 \def\@OrdinalstringF{%
5401   \fc@multiling{Ordinalstring}{F}%
5402 }%
```

The neuter version of \Ordinalstring:

```
5403 \def\@OrdinalstringN{%
5404   \fc@multiling{Ordinalstring}{N}%
5405 }%
```

Make \fmtord language dependent:

```
5406 \let\fmtord\fc@ord@multiling
5407 }
```

Check to see if babel, polyglossia or ngerman packages have been loaded, and if yes set fmtcount in multiling.

```
5408 \expandafter\@ifpackageloaded
5409 \expandafter{\ifxetex polyglossia\else babel\fi}%
5410 {%
5411   \@set@multiling@fmtcount
5412 }%
5413 {%
5414   \@ifpackageloaded{ngerman}%
5415   {%
5416     \FCloadlang{ngerman}%
5417     \@set@multiling@fmtcount
5418   }%
5419   {%
```

In the case that neither babel/polyglossia, nor ngerman has been loaded, then we go to multiling if a language has been loaded by package option, and to default language otherwise.

```
5420   \iffmtcount@language@option
5421     \@set@multiling@fmtcount
```

Some sanity check at the beginning of document may help the end user understand what is wrong:

```
5422   \AtBeginDocument{%
5423     \ifcsundef{languagename}%
5424     {%
5425       \PackageWarning{fmtcount}{%
5426         ‘\protect{languagename}’ is undefined, you should use package babel/polyglossia
5427         language via package option. Reverting to default language.
5428       }%
5429       \@setdef@ultfmtcount
5430     }{%
5431       \FC@iflangloaded{\languagename}{}%
```

The current \languagename is not a language that has been previously loaded. The correction is to have \languagename let to \fc@mainlang. Please note

that, as `\iffmtcount@language@option` is true, we know that `fmtcount` has loaded some language.

```
5432         \PackageWarning{fmtcount}{%
5433             Setting ‘\protect\languagename’ to ‘\fc@mainlang’.\MessageBreak
5434             Reason is that ‘\protect\languagename’ was ‘\languagename’,\MessageBreak
5435             but ‘\languagename’ was not loaded by fmtcount,\MessageBreak
5436             whereas ‘\fc@mainlang’ was the last language loaded by fmtcount ;
5437         }%
5438         \let\languagename\fc@mainlang
5439     }
5440 }%
5441 }
5442 \else
5443     \setdef@ultfmtcount
5444 \fi
5445 }%
5446 }

5447 \AtBeginDocument{%
5448     \ifcsundef{FBsupR}{\let\fc@textsuperscript\textsuperscript}{\let\fc@textsuperscript\fup}
5449 }
```

Backwards compatibility:

```
5450 \let\@ordinal=\@ordinalM
5451 \let\@ordinalstring=\@ordinalstringM
5452 \let\@Ordinalstring=\@OrdinalstringM
5453 \let\@numberstring=\@numberstringM
5454 \let\@Numberstring=\@NumberstringM
```