

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

Nicola L.C. Talbot

Vincent Belaïche

www.dickimaw-books.com

2014-07-18 (version 3.00)

Contents

1	Introduction	2
2	Available Commands	2
3	Package Options	8
4	Multilingual Support	8
4.1	Options for French	9
4.2	Prefixes	15
5	Configuration File <code>fmtcount.cfg</code>	15
6	LaTeX2HTML style	15
7	Acknowledgements	15
8	Troubleshooting	16
9	The Code	16
9.1	<code>fcnumparser.sty</code>	16
9.2	<code>fcprefix.sty</code>	26
9.3	<code>fmtcount.sty</code>	36
9.4	Multilanguage Definitions	62
9.4.1	<code>fc-american.def</code>	65
9.4.2	<code>fc-british.def</code>	66
9.4.3	<code>fc-english.def</code>	66
9.4.4	<code>fc-francais.def</code>	77
9.4.5	<code>fc-french.def</code>	77
9.4.6	<code>fc-frenchb.def</code>	107
9.4.7	<code>fc-german.def</code>	108

9.4.8	<code>fc-germanb.def</code>	118
9.4.9	<code>fc-italian</code>	118
9.4.10	<code>fc-ngerman.def</code>	119
9.4.11	<code>fc-ngermanb.def</code>	120
9.4.12	<code>fc-portuges.def</code>	120
9.4.13	<code>fc-portuguese.def</code>	135
9.4.14	<code>fc-spanish.def</code>	136
9.4.15	<code>fc-UKenglish.def</code>	153
9.4.16	<code>fc-USenglish.def</code>	154

1 Introduction

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

2 Available Commands

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

`\ordinal`

```
\ordinal{<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 `mem-oir`'s version of that command.

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

`\storeordinal` `\storeordinal{<label>}{<counter>}[<gender>]`

`\storeordinalstring` `\storeordinalstring{<label>}{<counter>}[<gender>]`

`\storeOrdinalstring` `\storeOrdinalstring{<label>}{<counter>}[<gender>]`

`\storeORDINALstring` `\storeORDINALstring{<label>}{<counter>}[<gender>]`

`\storenumberstring` `\storenumberstring{<label>}{<counter>}[<gender>]`

`\storeNumberstring` `\storeNumberstring{<label>}{<counter>}[<gender>]`

`\storeNUMBERstring` `\storeNUMBERstring{<label>}{<counter>}[<gender>]`

`\storeordinalnum` `\storeordinalnum{<label>}{<number>}[<gender>]`

`\storeordinalstringnum` `\storeordinalstring{<label>}{<number>}[<gender>]`

`\storeOrdinalstringnum` `\storeOrdinalstringnum{<label>}{<number>}[<gender>]`

`\storeORDINALstringnum` `\storeORDINALstringnum{<label>}{<number>}[<gender>]`

`\storenumberstringnum` `\storenumberstring{<label>}{<number>}[<gender>]`

\storeNumberstringnum	<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>\binarynum{<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`.

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 § 4.1.

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

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

4.1 Options for French

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

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

`fmtcountsetoptions`

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

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

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

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

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

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

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

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

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

`swiss` septante pour 70, huitante⁴ pour 80, et nonante pour 90

Il est à noter que la variante `belgian` est parfaitement correcte pour les francophones français⁵, et qu’elle est également utilisée en Suisse Romande hormis dans les cantons de Vaud, du Valais et de Fribourg. En ce qui concerne le mot “octante”, il n’est actuellement pas pris en charge et n’est guère plus utilisé, ce qui est sans doute dommage car il est sans doute plus acceptable que le “huitante” de certains de nos amis suisses.

`abbr` `\fmtcountsetoptions{abbr=<boolean>}`

L’option générale `abbr` permet de changer l’effet de `\ordinal`. Selon *<boolean>* on a :

`true` pour produire des ordinaux de la forme 2^e, ou

`false` pour produire des ordinaux de la forme 2^{eme} (par défaut)

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

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

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

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

⁴voir [Octante et huitante](#) sur le site d’Alain Lassine

⁵je précise que l’auteur de ces lignes est français

n-illiard plural

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

all plural

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

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

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

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

Le paramètre `<french plural control>` peut prendre les valeurs suivantes :

traditional	pour sélectionner la règle en usage chez les adultes à la date de parution de ce document, et dans le cas des numéraux cardinaux, lorsqu'ils ont une valeur cardinale,
reformed	pour suivre toute nouvelle recommandation à la date de parution de ce document, , et dans le cas des numéraux cardinaux, lorsqu'ils ont une valeur cardinale, l'idée des options traditional et reformed est donc de pouvoir contenter à la fois les anciens et les modernes, mais à dire vrai à la date où ce document est écrit elles ont exactement le même effet,
traditional o	pareil que traditional mais dans le cas des numéraux cardinaux, lorsqu'ils ont une valeur ordinale,
reformed o	pareil que reformed mais dans le cas des numéraux cardinaux, lorsqu'ils ont une valeur ordinale, de même que précédemment reformed o et traditional o ont exactement le même effet,
always	pour marquer toujours le pluriel, ceci n'est correct que pour « mil » vis à vis des règles en vigueur,
never	pour ne jamais marquer le pluriel, ceci est incorrect vis à vis des règles d'orthographe en vigueur,
multiple	pour marquer le pluriel lorsque le nombre considéré est multiplié par au moins 2, ceci est la règle en vigueur pour les nombres de la forme $\langle n \rangle$ illion et $\langle n \rangle$ illiard lorsque le nombre a une valeur cardinale,
multiple g-last	pour marquer le pluriel lorsque le nombre considéré est multiplié par au moins 2 est est <i>globalement</i> en dernière position, où "globalement" signifie qu'on considère le nombre formaté en entier, ceci est incorrect vis à vis des règles d'orthographe en vigueur,
multiple l-last	pour marquer le pluriel lorsque le nombre considéré est multiplié par au moins 2 et est <i>localement</i> en dernière position, où "localement" signifie qu'on considère seulement la portion du nombre qui multiplie soit l'unité, soit un $\langle n \rangle$ illion ou un $\langle n \rangle$ illiard ; ceci est la convention en vigueur pour le pluriel de "vingt" et de "cent" lorsque le nombre formaté a une valeur cardinale,

multiple lng-last pour marquer le pluriel lorsque le nombre considéré est multiplié par au moins 2 et est *localement* mais *non globalement* en dernière position, où “localement” et *globalement* on la même signification que pour les options multiple g-last et multiple l-last ; ceci est la convention en vigueur pour le pluriel de “vingt” et de “cent” lorsque le nombre formaté a une valeur ordinale,

multiple ng-last pour marquer le pluriel lorsque le nombre considéré est multiplié par au moins 2, et *n*’est pas *globalement* en dernière position, où “globalement” a la même signification que pour l’option multiple g-last ; ceci est la règle que j’infière être en vigueur pour les nombres de la forme $\langle n \rangle$ illion et $\langle n \rangle$ illiard lorsque le nombre a une valeur ordinale, mais à dire vrai pour des nombres aussi grands, par exemple « deux millions », je pense qu’il n’est tout simplement pas d’usage de dire « l’exemplaire deux million(s ?) » pour « le deux millionième exemplaire ».

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

$\langle x \rangle$ dans “ $\langle x \rangle$ plural”	traditional	reformed	traditional o	reformed o
vingt	multiple l-last		multiple lng-last	
cent	multiple l-last		multiple lng-last	
mil	always			
n-illion	multiple		multiple ng-last	
n-illiard	multiple		multiple ng-last	

Les configurations qui respectent les règles d’orthographe sont les suivantes :

- `\fmtcountsetoptions{french={all plural=reformed o}}` pour formater les numéraux cardinaux à valeur ordinale,
- `\fmtcountsetoptions{french={mil plural=multiple}}` pour activer l’alternance mil/mille.
- `\fmtcountsetoptions{french={all plural=reformed}}` pour revenir dans la configuration par défaut.

dash or space

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

Avant la réforme de l’orthographe de 1990, on ne met des traits d’union qu’entre les dizaines et les unités, et encore sauf quand le nombre n considéré est tel que $n \bmod 10 = 1$, dans ce cas on écrit “et un” sans trait d’union. Après la réforme de 1990, on recommande de mettre des traits d’union de partout sauf

autour de “mille”, “million” et “milliard”, et les mots analogues comme “billion”, “billiard”. Cette exception a toutefois été contestée par de nombreux auteurs, et on peut aussi mettre des traits d’union de partout. Mettre l’option *⟨dash or space⟩* à :

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

Par défaut, l’option vaut *reformed*.

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

L’option *scale* permet de configurer l’écriture des grands nombres. Mettre *⟨scale⟩* à :

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

Par défaut, l’option vaut *recursive*.

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

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

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

mil plural mark `\fmtcountsetoptions{french={mil plural mark=⟨any text⟩}}`

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

que « blini » veut dire « crêpes » (au pluriel).

4.2 Prefixes

`\latinnumeralstring`

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

`\latinnumeralstringnum`

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

5 Configuration File `fmtcount.cfg`

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

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

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

and if `fmtcount.cfg` has the line:

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

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

6 LaTeX2HTML style

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

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

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

This, I agree, is an unpleasant cludge.

7 Acknowledgements

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

8 Troubleshooting

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

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

9 The Code

9.1 fcnumparser.sty

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

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

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

`\fc@number@analysis`

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

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

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

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

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

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

No comma either, so fractional part is set empty.

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

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

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

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

```

22     \def\@tempb##1.\fc@end{\def\fc@fractional@part{##1}}%
23     \expandafter\@tempb\@tempa
24 \fi
25 }

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

Now, split `\fc@integer@part` into a sequence of `\fc@digit@<n>` with `<n>`

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

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

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

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

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

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

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

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

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

Here is the loop body:

```
82 {%
83 \count0 \@temp_i
84 \advance\count0 by \fc@digit@counter
85 \expandafter\def\expandafter\@tempd\expandafter{\csname fc@digit@\the\count0\endcsname}
86 \expandafter\def\expandafter\@tempe\expandafter{\csname fc@digit@\the\fc@digit@counter\endcsname}
87 \def\@tempa####1####2{\def\@tempb{\let####1####2}}%
88 \expandafter\expandafter\expandafter\@tempa\expandafter\@tempe\@tempd
89 \expandafter
90 }\@tempb
91 \advance\fc@digit@counter by 1 %
```

```

92     \fi
93     \next
94 }%
95 \let\next\@templ
96 \@templ

```

Split `\fc@fractional@part` into a sequence of `\fc@digit@<n>` with $\langle n \rangle$ ranging from `\fc@unit@weight - 1` to `\fc@min@weight` by step of -1 . This is much more simpler because we get the digits with the final range of index, so no post-processing loop is needed.

```

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

```

`\fc@parse@integer@digits`

```

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

```

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

`\fc@read@unit`

`\fc@read@unit` just reads one digit and form an integer in the range $[0..9]$.

First we check that the macro is not yet defined.

```

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

```

Arguments as follows:

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

```

121 \def\fc@read@unit#1#2{%
122 \ifnum#2>\fc@max@weight
123 #1=0\relax
124 \else

```

```

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

```

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

```

139 \ifcsundef{fc@read@hundred}{}%
140 \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro ‘fc@read@hundred’}

```

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

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

```

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

```

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

```

155 \ifcsundef{fc@read@thousand}{}%
156 \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro
157     ‘fc@read@thousand’}

```

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

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

```

158 \def\fc@read@thousand#1#2{%
159     {%
160         \fc@read@unit{\count0}{#2}%
161         \def\@tempa##1{\fc@read@hundred{\count1}{##1}}%

```

```

162 \count2=#2%
163 \advance\count2 by 1 %
164 \expandafter\@tempa{\the\count2}%
165 \multiply\count1 by 10 %
166 \advance\count1 by \count0 %
167 \def\@tempa##1{\def\@tempb{#1=##1\relax}}
168 \expandafter\@tempa\expandafter{\the\count1}%
169 \expandafter
170 }\@tempb
171 }

```

`\fc@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.

```

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

```

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

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

```

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

```

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

```

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

```

Arguments as follows:

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

Actually `\fc@check@nonzeros` is just a wrapper to collect arguments, and the

real job is delegated to `\fc@@check@nonzeros@inner` which is called inside a group.

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

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

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

Then we do the real job

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

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

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

`\fc@@check@nonzeros@inner` Macro `\fc@@check@nonzeros@inner` Check whether some part of the parsed value contains some non-zero digit At the call of this macro we expect that:

```
\@tempa  input/output macro:
          input   minimum unit weight at which start to search the
                   non-zeros
          output  macro may have been redefined
\@tempb  input/output macro:
          input   maximum unit weight at which end to search the non-
                   zeros
          output  macro may have been redefined
\@tempc  output macro: 0 if all-zeros, 1 if at least one zero is found
\count0  output counter: weight + 1 of the first found non zero starting from
          minimum weight.
```

```
204 \def\fc@@check@nonzeros@inner{%
205   \ifnum\count0<\fc@min@weight
206     \count0=\fc@min@weight\relax
207   \fi
208   \ifnum\count1>\fc@max@weight\relax
209     \count1=\fc@max@weight
210   \fi
211   \count2\count0 %
212   \advance\count2 by 1 %
213   \ifnum\count0>\count1 %
214     \PackageError{fcnumparser}{Unexpected arguments}{Number in argument 2 of macro
215       'fc@check@nonzeros' must be at least equal to number in argument 1}%
216   \else
217     \fc@@check@nonzeros@inner@loopbody
218   \ifnum\@tempc>0 %
```

```

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

```

`\fc@intpart@find@last`

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

```

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

```

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

```

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

```

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

```

257     \count0=\fc@unit@weight\space

```

```

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

```

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

```

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

```

Now propagate result after closing bracket into counter #1.

```

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

some \fc@case. First we define \@tempa to split \the\toks1 between parts before and after some potential \fc@case.

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

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

```
293 \toksdef\@tempt0 %
```

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

```
294 \def\@templ{%
295 \expandafter\@tempa\the\toks1 \fc@end
296 \ifx\@tempb\@empty
```

\@tempb empty means that the only \fc@case found in \the\toks1 is the dummy one. So we end the loop here, \toks2 contains the last word.

```
297 \let\next\relax
298 \else
```

\@tempb is not empty, first we use

```
299 \expandafter\expandafter\expandafter\@tempt
300 \expandafter\expandafter\expandafter{%
301 \expandafter\the\expandafter\@tempt
302 \expandafter\fc@case\the\toks2}%
303 \toks1\toks3 %
304 \fi
305 \next
306 }%
307 \let\next\@templ
308 \@templ
309 \edef\@tempa{\def\noexpand#2{\the\toks0}\def\noexpand#3{\the\toks2}}%
310 \expandafter
311 }\@tempa
312 }
```

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

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

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

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

```
317 \toks1{#1}%
318 \toks0{ }%
```

```

319 \toksdef\@tempt0 %
We define \@tempa in order to pop the first letter from the remaining of word.
320 \def\@tempa##1##2\fc@nil{%
321 \toks2{##1}%
322 \toks3{##2}%
323 \def\@tempb{##2}%
324 }%

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

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

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

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

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

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

```

9.2 fcprefix.sty

Pseudo-latin prefixes.

```

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

```

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

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

```

349 \define@key{fcprefix}{use duode and unde}[below20]{%
350 \ifthenelse{\equal{#1}{below20}}{%
351 \def\fc@duodeandunde{2}%

```

```

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

```

Default is ‘below 20’ like in French.

```

361 \def\fc@duodeandunde{2}

```

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

```

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

```

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

```

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

```

Default is to set accute accent like in French.

```

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

```

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

arabic for which millia squared is noted as ‘millia²’

prefix for which millia squared is noted as ‘bismillia’

```

387 \define@key{fcprefix}{power of millia}[prefix]{%

```

```

388 \ifthenelse{\equal{#1}{prefix}}{%
389     \let\fc@power@of@millia@init@gobbletwo
390     \let\fc@power@of@millia\fc@@prefix@millia
391 }{%
392     \ifthenelse{\equal{#1}{arabic}}{%
393         \let\fc@power@of@millia@init@gobbletwo
394         \let\fc@power@of@millia\fc@@arabic@millia
395     }{%
396         \ifthenelse{\equal{#1}{recursive}}{%
397             \let\fc@power@of@millia@init\fc@@recurse@millia@init
398             \let\fc@power@of@millia\fc@@recurse@millia
399         }{%
400             \PackageError{fc@prefix}{Unexpected option}{%
401                 Option ‘power of millia’ expects ‘recursive’, ‘arabic’, or ‘prefix’ }%
402             }%
403         }%
404     }%
405 }

```

Arguments as follows:

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

```

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```
#1 output macro
#2 number with current weight  $w$ 
425 \def\fc@@arabic@millia#1#2{%
426   \ifnnum#2=0 %
427     \let#1\@empty
428   \else
429     \edef#1{millia\^{\the#2}}%
430   \fi
431 }
```

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

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

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

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

`\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.

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

Arguments as follows:

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

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

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

Now parse number from expanded input.

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

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

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

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

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

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

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

Loop body. Here we read a group of 3 consecutive digits $d_2d_1d_0$ and place them respectively into `\count3`, `\count4`, and `\count5`.

```
451   \fc@read@unit{\count3}{\count2}%
452   \advance\count2 by 1 %
453   \fc@read@unit{\count4}{\count2}%
454   \advance\count2 by 1 %
455   \fc@read@unit{\count5}{\count2}%
456   \advance\count2 by 1 %
```

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

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

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

```
466   \let\@temp\@tempa
467   \edef\@tempa{%
```

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

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

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

```
479     \ifnum\count4=0 %
480     % x0(0..9)
481     \ifnum\count2=3 %
482     % Absolute weight zero
483     \ifcase\count3 \@tempt
```

```

484         \or m%
485         \or b%
486         \or tr%
487         \or quadr%
488         \or quin\@tempt
489         \or sex\@tempt
490         \or sep\@tempt
491         \or oc\@tempt
492         \or non%
493         \fi
494     \else

```

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

```

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

```

```

531         \fi
Insert the millia(n÷3) only if  $d_2d_1d_0 \neq 0$ , i.e. if one of \count3 \count4 or
\count5 is non zero.
532         \@tempm
And append previous version of \@tempa.
533         \@temp
534     }%
```

“Concatenate” millia to \@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’.

```

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

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

```

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

latin@numeral@prefix 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.

```

547 \ifcsundef{fc@latin@numeral@prefix}{-}{%
548   \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
549     ‘fc@latin@numeral@prefix’}}
```

Arguments as follows:

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

```

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

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

```

556     \let\@tempm\@empty
```

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

```

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

```

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

```

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

```

Check the use of duodevicies instead of octodecies.

```

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

```

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

```

599     \ifcase\count3 %
600     \or

```

```

601         \ifnum\count2=3 %
602             s\fc@prefix@eacute emel%
603         \else
604             \ifnum\count2>\fc@max@weight\else un\fi
605         \fi
606     \or bis%
607     \or ter%
608     \or quater%
609     \or quinquies%
610     \or sexies%
611     \or septies%
612     \or octies%
613     \or novies%
614     \fi
615 \else

```

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

```

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

```

Concatenate millies to \@tempm so that it is equal to millies^{n÷3} at the next iteration. Here we just have plain concatenation, contrary to cardinal for which

a prefix can be used instead.

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

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

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

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

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

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

```

673 \def\fc@call@opt@arg@first#1{%
674   \def\@tempb{%
675     \ifx[\@tempa
676       \def\@tempc[####1]###2{{#1{####1}{###2}}}%
677     \else
678       \def\@tempc####1{{#1}{####1}}}%
679     \fi
680   \@tempc
681 }%
682 \futurelet\@tempa
683 \@tempb
684 }
685
686 \let\fc@call\fc@call@opt@arg@first

```

User API.

innumeralstringnum

Macro \@latinnumeralstringnum. Arguments as follows:

```

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

```

Arguments as follows:

```

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

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

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

```

9.3 fmtcount.sty

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

```

705 \NeedsTeXFormat{LaTeX2e}
706 \ProvidesPackage{fmtcount}[2014/07/18 v3.00]
707 \RequirePackage{ifthen}

```

```
708 \RequirePackage{keyval}
709 \RequirePackage{etoolbox}
710 \RequirePackage{fcprefix}
```

```
711 \RequirePackage{ifxetex}
```

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

```
712 \RequirePackage{amsgen}
```

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

`\fmtord`

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

`\padzeroes`

```
\padzeroes[n]
```

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

```
714 \newcount\c@padzeroesN
```

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

```
716 \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.

```
717 \newcount\fc@tmpcatcode
```

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

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

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

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

```
722 {%
```

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

```
724 \catcode '\@ 11 \relax
```

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

```
726 {%
```

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

```
728 {%
```

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

```
730 }%
```

```
731 }%
```

```

732     \gappto\fc@languages{,#1}%
733     }%
734     \gdef\fc@mainlang{#1}%
735     }%
736     {}%
737     \catcode ‘\@ \fc@tmpcatcode\relax
738     }%
739 }

```

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

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

```

740 \newcommand{\FC@iflangloaded}[3]{%
741   \ifcsundef{ver@fc-#1.def}{#3}{#2}%
742 }

```

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

```

743 \newcommand*{\ProvidesFCLanguage}[1]{%
744   \ProvidesFile{fc-#1.def}%
745 }

```

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

```

746 \newif\iffntcount@language@option
747 \fntcount@language@optionfalse

```

`\supported@language@list` Declare list of supported languages, as a comma separated list. No space, no empty items. Each item is a language for which `fntcount` is able to load language specific definitions. 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 $\LaTeX 2_{\epsilon}$.

```

748 \newcommand*\fc@supported@language@list{%
749   english,%
750   UKenglish,%
751   british,%

```

```

752 USenglish,%
753 american,%
754 spanish,%
755 portuges,%
756 french,%
757 frenchb,%
758 francais,%
759 german,%
760 germanb,%
761 ngerman,%
762 ngermanb,%
763 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 :

```

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

```

That case should never happen !

```

766     \PackageError{fmtcount}{Macro ‘\protect\fc@iterate@on@languages’ is empty}{You should :
767       Something is broken within \texttt{fmtcount}, please report the issue on
768       \texttt{https://github.com/search?q=fmtcount\&ref=cmdform\&type=Issues}}%
769   \else
770     \let\fc@iterate@on@languages@body#1
771     \expandafter\@fc@iterate@on@languages\fc@supported@language@list,\@nil,%
772   \fi
773 }
774 \def\@fc@iterate@on@languages#1,{%
775   {%
776     \def\@tempa{#1}%
777     \ifx\@tempa\@nnil
778       \let\@tempa\@empty
779     \else
780       \def\@tempa{%
781         \fc@iterate@on@languages@body{#1}%
782         \@fc@iterate@on@languages
783       }%
784     \fi
785     \expandafter
786   }\@tempa
787 }%

```

babelorpolyglossialdf

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

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

- `babel` language definition file `⟨language⟩.ldf` has been loaded — conditionally to compilation with `latex`, not `xelatex`.
- `polyglossia` language definition file `gloss-⟨language⟩.ldf` has been loaded — conditionally to compilation with `xelatex`, not `latex`.
- `⟨language⟩` option has been passed to package `fmtcount`.

```

788 \newcommand*{\@fc@loadifbabelorpolyglossialdf}[1]{%
789   \ifxetex
790     \IfFileExists{gloss-#1.ldf}{\ifcsundef{#1@loaded}}{\FCloadlang{#1}}{}%
791   \else
792     \ifcsundef{ver@#1.ldf}{\FCloadlang{#1}}%
793   \fi
794 }
```

Load appropriate language definition files:

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

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

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

`french`

```

797 \define@key{fmtcount}{french}[france]{%
798   \@FC@iflangloaded{french}%
799   {%
800     \setkeys{french}{#1}%
801   }%
802   {%
803     \PackageError{fmtcount}%
804     {Language ‘french’ not defined}%
805     {You need to load babel before loading fmtcount}%
806   }%
807 }
```

`fmtord` Key to determine how to display the ordinal

```

808 \define@key{fmtcount}{fmtord}{%
809   \ifthenelse{\equal{#1}{level}
810             \or\equal{#1}{raise}
811             \or\equal{#1}{user}}%
812   {%
813     \def\fmtcount@fmtord{#1}%
814   }%
815   {%
```

```

816 \PackageError{fmtcount}%
817 {Invalid value '#1' to fmtord key}%
818 {Option 'fmtord' can only take the values 'level', 'raise'
819 or 'user'}%
820 }%
821 }

```

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

```

822 \newif\iffmtord@abbrv
823 \fmtord@abbrvfalse
824 \define@key{fmtcount}{abbrv}[true]{%
825 \ifthenelse{\equal{#1}{true}\or\equal{#1}{false}}%
826 {%
827 \csname fmtord@abbrv#1\endcsname
828 }%
829 {%
830 \PackageError{fmtcount}%
831 {Invalid value '#1' to fmtord key}%
832 {Option 'fmtord' can only take the values 'true' or
833 'false'}%
834 }%
835 }

```

`prefix`

```

836 \define@key{fmtcount}{prefix}[scale=long]{%
837 \RequirePackage{fmtprefix}%
838 \fmtprefixsetoption{#1}%
839 }

```

`\fmtcountsetoptions` Define command to set options.

```

840 \newcommand*{\fmtcountsetoptions}[1]{%
841 \def\fmtcount@fmtord{}%
842 \setkeys{fmtcount}{#1}%
843 \@FC@iflangloaded{french}{\ifcsundef{@ordinalstringMfrench}}%
844 {%
845 \edef\@ordinalstringMfrench{\noexpand
846 \csname @ordinalstringMfrench\fmtcount@french\noexpand\endcsname}%
847 \edef\@ordinalstringFfrench{\noexpand
848 \csname @ordinalstringFfrench\fmtcount@french\noexpand\endcsname}%
849 \edef\@OrdinalstringMfrench{\noexpand
850 \csname @OrdinalstringMfrench\fmtcount@french\noexpand\endcsname}%
851 \edef\@OrdinalstringFfrench{\noexpand
852 \csname @OrdinalstringFfrench\fmtcount@french\noexpand\endcsname}%
853 \edef\@numberstringMfrench{\noexpand
854 \csname @numberstringMfrench\fmtcount@french\noexpand\endcsname}%
855 \edef\@numberstringFfrench{\noexpand
856 \csname @numberstringFfrench\fmtcount@french\noexpand\endcsname}%
857 \edef\@NumberstringMfrench{\noexpand

```

```

858     \csname @NumberstringMfrench\fmtcount@french\noexpand\endcsname}%
859     \edef\@NumberstringFfrench{\noexpand
860     \csname @NumberstringFfrench\fmtcount@french\noexpand\endcsname}%
861   }{}{}%
862   \ifthenelse{\equal{\fmtcount@fmtord}{level}}%
863   {%
864     \renewcommand{\fmtord}[1]{##1}%
865   }%
866   {%
867     \ifthenelse{\equal{\fmtcount@fmtord}{raise}}%
868     {%
869       \renewcommand{\fmtord}[1]{\textsuperscript{##1}}%
870     }%
871     {%
872     }%
873   }
874 }

```

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.

```

875 \InputIfFileExists{fmtcount.cfg}%
876 {%
877   \PackageInfo{fmtcount}{Using configuration file fmtcount.cfg}%
878 }%
879 {%
880 }

```

\metallanguage

```

881 \newcommand*\@fc@declare@language@option[1]{%
882   \DeclareOption{#1}{%
883     \@FC@iflangloaded{#1}{}%
884     \fmtcount@language@optiontrue
885     \FCloadlang{#1}%
886   }}%
887 \fc@iterate@on@languages\@fc@declare@language@option

```

level

```

888 \DeclareOption{level}{\def\fmtcount@fmtord{level}%
889   \def\fmtord#1{#1}}

```

raise

```

890 \DeclareOption{raise}{\def\fmtcount@fmtord{raise}%
891   \def\fmtord#1{\textsuperscript{#1}}}

```

Process package options

```

892 \ProcessOptions\relax

```

\@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.)

```
893 \newcount\@DT@modctr
894 \newcommand*\@FCmodulo}[2]{%
895   \@DT@modctr=#1\relax
896   \divide \@DT@modctr by #2\relax
897   \multiply \@DT@modctr by #2\relax
898   \advance #1 by -\@DT@modctr
899 }
```

The following registers are needed by \@ordinal etc

```
900 \newcount\@ordinalctr
901 \newcount\@orgargctr
902 \newcount\@strctr
903 \newcount\@tmpstrctr
```

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

```
904 \newif\if@DT@padzeroes
905 \newcount\@DT@loopN
906 \newcount\@DT@X
```

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

```
907 \newcommand*\@binary}[1]{%
908   \@DT@padzeroestrue
909   \@DT@loopN=17\relax
910   \@strctr=\@DT@loopN
911   \whiledo{\@strctr<\c@padzeroesN}{\advance \@strctr by 1}%
912   \@strctr=65536\relax
913   \@DT@X=#1\relax
914   \loop
915     \@DT@modctr=\@DT@X
916     \divide \@DT@modctr by \@strctr
917     \ifthenelse{\boolean{@DT@padzeroes}
918       \and \(\@DT@modctr=0\)}
919       \and \(\@DT@loopN>\c@padzeroesN\)}%
920     {}%
921     {\the \@DT@modctr}%
922     \ifnum \@DT@modctr=0\else \@DT@padzeroesfalse\fi
923     \multiply \@DT@modctr by \@strctr
924     \advance \@DT@X by -\@DT@modctr
925     \divide \@strctr by 2\relax
926     \advance \@DT@loopN by -1\relax
927   \ifnum \@strctr>1
928     \repeat
929     \the \@DT@X
930 }
931
```

```
932 \let\binarynum=\@binary
```

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

```
933 \newcommand*\@octal}[1]{%
934   \ifnum#1>32768
935     \PackageError{fmtcount}%
936       {Value of counter too large for \protect\@octal}
937       {Maximum value 32768}
938   \else
939     \@DT@padzeroestruer
940     \@DT@loopN=6\relax
941     \@strctr=\@DT@loopN
942     \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by 1}%
943     \@strctr=32768\relax
944     \@DT@X=#1\relax
945     \loop
946       \@DT@modctr=\@DT@X
947       \divide\@DT@modctr by \@strctr
948       \ifthenelse{\boolean{@DT@padzeroes}
949         \and \(\@DT@modctr=0\}
950         \and \(\@DT@loopN>\c@padzeroesN\)}%
951         {\the\@DT@modctr}%
952         \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
953         \multiply\@DT@modctr by \@strctr
954         \advance\@DT@X by -\@DT@modctr
955         \divide\@strctr by 8\relax
956         \advance\@DT@loopN by -1\relax
957     \ifnum\@strctr>1
958     \repeat
959     \the\@DT@X
960   \fi
961 }
962 \let\octalnum=\@octal
```

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

```
963 \newcommand*\@hexadecimal}[1]{%
964   \ifcase#1\or1\or2\or3\or4\or5\or
965   6\or7\or8\or9\or a\or b\or c\or d\or e\or f\fi
966 }
```

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

```
967 \newcommand*\@hexadecimal}[1]{%
968   \@DT@padzeroestruer
969   \@DT@loopN=5\relax
970   \@strctr=\@DT@loopN
971   \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by 1}%
972   \@strctr=65536\relax
973   \@DT@X=#1\relax
```

```

974 \loop
975   \@DT@modctr=\@DT@X
976   \divide\@DT@modctr by \@strctr
977   \ifthenelse{\boolean{@DT@padzeroes}
978     \and \(\@DT@modctr=0\}
979     \and \(\@DT@loopN>\c@padzeroesN\)}
980     {\@hexadecimal\@DT@modctr}%
981     \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
982     \multiply\@DT@modctr by \@strctr
983     \advance\@DT@X by -\@DT@modctr
984     \divide\@strctr by 16\relax
985     \advance\@DT@loopN by -1\relax
986   \ifnum\@strctr>1
987     \repeat
988     \@hexadecimal\@DT@X
989 }
990 \let\hexadecimalnum=\@hexadecimal

```

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

```

991 \newcommand*{\@Hexadecimal}[1]{%
992   \ifcase#1\or1\or2\or3\or4\or5\or6\or
993   7\or8\or9\or A\or B\or C\or D\or E\or F\fi
994 }

```

`\Hexadecimalnum` Uppercase hexadecimal

```

995 \newcommand*{\@Hexadecimal}[1]{%
996   \@DT@padzeroestrue
997   \@DT@loopN=5\relax
998   \@strctr=\@DT@loopN
999   \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by 1}%
1000  \@strctr=65536\relax
1001  \@DT@X=#1\relax
1002  \loop
1003    \@DT@modctr=\@DT@X
1004    \divide\@DT@modctr by \@strctr
1005    \ifthenelse{\boolean{@DT@padzeroes}
1006      \and \(\@DT@modctr=0\}
1007      \and \(\@DT@loopN>\c@padzeroesN\)}%
1008      {\@Hexadecimal\@DT@modctr}%
1009      \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
1010      \multiply\@DT@modctr by \@strctr
1011      \advance\@DT@X by -\@DT@modctr
1012      \divide\@strctr by 16\relax
1013      \advance\@DT@loopN by -1\relax
1014    \ifnum\@strctr>1
1015      \repeat
1016      \@Hexadecimal\@DT@X
1017 }
1018

```

```
1019 \let\Hexadecimalnum=\@Hexadecimal
```

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

```
1020 \newcommand*\@aaalph}[1]{%
1021   \@DT@loopN=#1\relax
1022   \advance\@DT@loopN by -1\relax
1023   \divide\@DT@loopN by 26\relax
1024   \@DT@modctr=\@DT@loopN
1025   \multiply\@DT@modctr by 26\relax
1026   \@DT@X=#1\relax
1027   \advance\@DT@X by -1\relax
1028   \advance\@DT@X by -\@DT@modctr
1029   \advance\@DT@loopN by 1\relax
1030   \advance\@DT@X by 1\relax
1031   \loop
1032     \@alph\@DT@X
1033     \advance\@DT@loopN by -1\relax
1034   \ifnum\@DT@loopN>0
1035     \repeat
1036 }
1037
1038 \let\aaalphnum=\@aaalph
```

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

```
1039 \newcommand*\@AAAAlph}[1]{%
1040   \@DT@loopN=#1\relax
1041   \advance\@DT@loopN by -1\relax
1042   \divide\@DT@loopN by 26\relax
1043   \@DT@modctr=\@DT@loopN
1044   \multiply\@DT@modctr by 26\relax
1045   \@DT@X=#1\relax
1046   \advance\@DT@X by -1\relax
1047   \advance\@DT@X by -\@DT@modctr
1048   \advance\@DT@loopN by 1\relax
1049   \advance\@DT@X by 1\relax
1050   \loop
1051     \@Alph\@DT@X
1052     \advance\@DT@loopN by -1\relax
1053   \ifnum\@DT@loopN>0
1054     \repeat
1055 }
1056
1057 \let\AAAalphnum=\@AAAAlph
```

\abalphnum Lowercase alphabetical representation

```
1058 \newcommand*\@abalph}[1]{%
1059   \ifnum#1>17576\relax
1060     \PackageError{fmtcount}%
1061     {Value of counter too large for \protect\@abalph}%
```

```

1062   {Maximum value 17576}%
1063   \else
1064     \@DT@padzeroestru
1065     \@strctr=17576\relax
1066     \@DT@X=#1\relax
1067     \advance\@DT@X by -1\relax
1068     \loop
1069       \@DT@modctr=\@DT@X
1070       \divide\@DT@modctr by \@strctr
1071       \ifthenelse{\boolean{@DT@padzeroes}
1072         \and \(\@DT@modctr=1\)}%
1073         {{{\@alph\@DT@modctr}%
1074         \ifnum\@DT@modctr=1\else\@DT@padzeroesfalse\fi
1075         \multiply\@DT@modctr by \@strctr
1076         \advance\@DT@X by -\@DT@modctr
1077         \divide\@strctr by 26\relax
1078         \ifnum\@strctr>1
1079         \repeat
1080         \advance\@DT@X by 1\relax
1081         \@alph\@DT@X
1082     \fi
1083 }
1084
1085 \let\abalphnum=\@abalph

```

\ABAlphnum Uppercase alphabetical representation

```

1086 \newcommand*{\@ABAlph}[1]{%
1087   \ifnum#1>17576\relax
1088     \PackageError{fmtcount}%
1089     {Value of counter too large for \protect\@ABAlph}%
1090     {Maximum value 17576}%
1091   \else
1092     \@DT@padzeroestru
1093     \@strctr=17576\relax
1094     \@DT@X=#1\relax
1095     \advance\@DT@X by -1\relax
1096     \loop
1097       \@DT@modctr=\@DT@X
1098       \divide\@DT@modctr by \@strctr
1099       \ifthenelse{\boolean{@DT@padzeroes}\and
1100         \(\@DT@modctr=1\)}{{\@Alph\@DT@modctr}%
1101       \ifnum\@DT@modctr=1\else\@DT@padzeroesfalse\fi
1102       \multiply\@DT@modctr by \@strctr
1103       \advance\@DT@X by -\@DT@modctr
1104       \divide\@strctr by 26\relax
1105       \ifnum\@strctr>1
1106       \repeat
1107       \advance\@DT@X by 1\relax
1108       \@Alph\@DT@X

```

```

1109 \fi
1110 }
1111
1112 \let\ABAlphnum=\@ABAlph

```

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

```

1113 \def\@fmtc@count#1#2\relax{%
1114   \if\relax#1%
1115   \else
1116     \advance\@strctr by 1\relax
1117     \@fmtc@count#2\relax
1118   \fi
1119 }

```

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

```

1120 \newcommand{\@decimal}[1]{%
1121   \@strctr=0\relax
1122   \expandafter\@fmtc@count\number#1\relax
1123   \@DT@loopN=\c@padzeroesN
1124   \advance\@DT@loopN by -\@strctr
1125   \ifnum\@DT@loopN>0\relax
1126     \@strctr=0\relax
1127     \whiledo{\@strctr < \@DT@loopN}{\advance\@strctr by 1\relax}%
1128   \fi
1129   \number#1\relax
1130 }
1131
1132 \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.

```

1133 \newcommand{\FCordinal}[1]{%
1134   \expandafter\protect\expandafter\ordinalnum{%
1135     \expandafter\the\csname c@#1\endcsname}%
1136 }

```

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

```
1137 \ifcsundef{ordinal}
1138 {\let\ordinal\FCordinal}%
1139 {%
1140   \PackageWarning{fmtcount}%
1141   {\protect\ordinal \space already defined use
1142    \protect\FCordinal \space instead.}
1143 }
```

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

```
1144 \newcommand*{\ordinalnum}[1]{%
1145   \new@ifnextchar[%
1146   {\@ordinalnum{#1}}%
1147   {\@ordinalnum{#1}[m]}%
1148 }
```

`\@ordinalnum` Display ordinal according to gender (neuter added in v1.1, `\xspace` added in v1.2, and removed in v1.3⁶):

```
1149 \def\@ordinalnum#1[#2]{%
1150   {%
1151     \ifthenelse{\equal{#2}{f}}%
1152     {%
1153       \protect\@ordinalF{#1}{\@fc@ordstr}%
1154     }%
1155     {%
1156       \ifthenelse{\equal{#2}{n}}%
1157       {%
1158         \protect\@ordinalN{#1}{\@fc@ordstr}%
1159       }%
1160       {%
1161         \ifthenelse{\equal{#2}{m}}%
1162         {}%
1163         {%
1164           \PackageError{fmtcount}%
1165           {Invalid gender option '#2'}%
1166           {Available options are m, f or n}%
1167         }%
1168         \protect\@ordinalM{#1}{\@fc@ordstr}%
1169       }%
1170     }%
1171     \@fc@ordstr
1172   }%
1173 }
```

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

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

```
1174 \newcommand*\storeordinal}[2]{%
1175   \expandafter\protect\expandafter\storeordinalnum{#1}{%
1176     \expandafter\the\csname c@#2\endcsname}%
1177 }
```

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

```
1178 \newcommand*\storeordinalnum}[2]{%
1179   \@ifnextchar[%
1180     {\@storeordinalnum{#1}{#2}}%
1181     {\@storeordinalnum{#1}{#2}[m]}%
1182 }
```

`\@storeordinalnum` Store ordinal according to gender:

```
1183 \def\@storeordinalnum#1#2[#3]{%
1184   \ifthenelse{\equal{#3}{f}}%
1185     {%
1186       \protect\@ordinalF{#2}{\@fc@ord}
1187     }%
1188     {%
1189       \ifthenelse{\equal{#3}{n}}%
1190         {%
1191           \protect\@ordinalN{#2}{\@fc@ord}%
1192         }%
1193         {%
1194           \ifthenelse{\equal{#3}{m}}%
1195             {}%
1196             {%
1197               \PackageError{fmtcount}%
1198                 {Invalid gender option ‘#3’}%
1199                 {Available options are m or f}%
1200             }%
1201           \protect\@ordinalM{#2}{\@fc@ord}%
1202         }%
1203     }%
1204   \expandafter\let\csname @fcs@#1\endcsname\@fc@ord
1205 }
```

`\FMCuse` Get stored information:

```
1206 \newcommand*\FMCuse}[1]{\csname @fcs@#1\endcsname}
```

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

```
1207 \newcommand*\ordinalstring}[1]{%
1208   \expandafter\protect\expandafter\ordinalstringnum{%
1209     \expandafter\the\csname c@#1\endcsname}%
1210 }
```

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

```
1211 \newcommand{\ordinalstringnum}[1]{%
1212   \new@ifnextchar[%
1213     {\@ordinal@string{#1}}%
1214     {\@ordinal@string{#1}[m]}}%
1215 }
```

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

```
1216 \def\@ordinal@string#1[#2]{%
1217   {%
1218     \ifthenelse{\equal{#2}{f}}%
1219     {%
1220       \protect\@ordinalstringF{#1}{\@fc@ordstr}%
1221     }%
1222     {%
1223       \ifthenelse{\equal{#2}{n}}%
1224       {%
1225         \protect\@ordinalstringN{#1}{\@fc@ordstr}%
1226       }%
1227       {%
1228         \ifthenelse{\equal{#2}{m}}%
1229         {}%
1230         {%
1231           \PackageError{fmtcount}%
1232             {Invalid gender option ‘#2’ to \protect\ordinalstring}%
1233             {Available options are m, f or n}%
1234         }%
1235         \protect\@ordinalstringM{#1}{\@fc@ordstr}%
1236       }%
1237     }%
1238     \@fc@ordstr
1239   }%
1240 }
```

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

```
1241 \newcommand*\storeordinalstring}[2]{%
1242   \expandafter\protect\expandafter\storeordinalstringnum{#1}{%
1243     \expandafter\the\csname c@#2\endcsname}%
1244 }
```

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

```
1245 \newcommand*\storeordinalstringnum}[2]{%
1246   \@ifnextchar[%
1247     {\@store@ordinal@string{#1}{#2}}%
1248     {\@store@ordinal@string{#1}{#2}[m]}}%
1249 }
```

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

```
1250 \def\@store@ordinal@string#1#2[#3]{%
1251   \ifthenelse{\equal{#3}{f}}{%
1252     {%
1253       \protect\@ordinalstringF{#2}{\@fc@ordstr}%
1254     }%
1255     {%
1256       \ifthenelse{\equal{#3}{n}}{%
1257         {%
1258           \protect\@ordinalstringN{#2}{\@fc@ordstr}%
1259         }%
1260         {%
1261           \ifthenelse{\equal{#3}{m}}{%
1262             {}%
1263           }%
1264           \PackageError{fmtcount}%
1265             {Invalid gender option ‘#3’ to \protect\ordinalstring}%
1266             {Available options are m, f or n}%
1267           }%
1268           \protect\@ordinalstringM{#2}{\@fc@ordstr}%
1269         }%
1270       }%
1271     \expandafter\let\csname @fcs@#1\endcsname\@fc@ordstr
1272 }
```

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

```
1273 \newcommand*\Ordinalstring[1]{%
1274   \expandafter\protect\expandafter\Ordinalstringnum{%
1275     \expandafter\the\csname c@#1\endcsname}%
1276 }
```

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

```
1277 \newcommand*\Ordinalstringnum[1]{%
1278   \new@ifnextchar[%
1279     {\@Ordinal@string{#1}}%
1280     {\@Ordinal@string{#1}[m]}%
1281 }
```

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

```
1282 \def\@Ordinal@string#1[#2]{%
1283   {%
1284     \ifthenelse{\equal{#2}{f}}{%
1285       {%
1286         \protect\@OrdinalstringF{#1}{\@fc@ordstr}%
1287       }%
1288       {%
```

```

1289     \ifthenelse{\equal{#2}{n}}%
1290     {%
1291     \protect\@OrdinalstringN{#1}{\@fc@ordstr}%
1292     }%
1293     {%
1294     \ifthenelse{\equal{#2}{m}}%
1295     {}%
1296     {%
1297     \PackageError{fmtcount}%
1298     {Invalid gender option ‘#2’}%
1299     {Available options are m, f or n}%
1300     }%
1301     \protect\@OrdinalstringM{#1}{\@fc@ordstr}%
1302     }%
1303     }%
1304     \@fc@ordstr
1305     }%
1306 }

```

`\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.

```

1307 \newcommand*\storeOrdinalstring}[2]{%
1308 \expandafter\protect\expandafter\storeOrdinalstringnum{#1}{%
1309 \expandafter\the\csname c@#2\endcsname}%
1310 }

```

`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.

```

1311 \newcommand*\storeOrdinalstringnum}[2]{%
1312 \@ifnextchar[%
1313 {\@store@Ordinal@string{#1}{#2}}%
1314 {\@store@Ordinal@string{#1}{#2}[m]}%
1315 }

```

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

```

1316 \def\@store@Ordinal@string#1#2[#3]{%
1317 \ifthenelse{\equal{#3}{f}}%
1318 {%
1319 \protect\@OrdinalstringF{#2}{\@fc@ordstr}%
1320 }%
1321 {%
1322 \ifthenelse{\equal{#3}{n}}%
1323 {%
1324 \protect\@OrdinalstringN{#2}{\@fc@ordstr}%
1325 }%
1326 {%

```

```

1327     \ifthenelse{\equal{#3}{m}}%
1328     {}%
1329     {%
1330         \PackageError{fmtcount}%
1331         {Invalid gender option ‘#3’}%
1332         {Available options are m or f}%
1333     }%
1334     \protect\@OrdinalstringM{#2}{\@fc@ordstr}%
1335 }%
1336 }%
1337 \expandafter\let\csname @fcs#1\endcsname\@fc@ordstr
1338 }

```

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

```

1339 \newcommand*{\storeORDINALstring}[2]{%
1340 \expandafter\protect\expandafter\storeORDINALstringnum{#1}{%
1341 \expandafter\the\csname c@#2\endcsname}%
1342 }

```

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

```

1343 \newcommand*{\storeORDINALstringnum}[2]{%
1344 \@ifnextchar[%
1345 {\@store@ORDINAL@string{#1}{#2}}%
1346 {\@store@ORDINAL@string{#1}{#2}[m]}%
1347 }

```

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

```

1348 \def\@store@ORDINAL@string#1#2[#3]{%
1349 \ifthenelse{\equal{#3}{f}}%
1350 {%
1351 \protect\@ordinalstringF{#2}{\@fc@ordstr}%
1352 }%
1353 {%
1354 \ifthenelse{\equal{#3}{n}}%
1355 {%
1356 \protect\@ordinalstringN{#2}{\@fc@ordstr}%
1357 }%
1358 {%
1359 \ifthenelse{\equal{#3}{m}}%
1360 {}%
1361 {%
1362 \PackageError{fmtcount}%
1363 {Invalid gender option ‘#3’}%
1364 {Available options are m or f}%
1365 }%
1366 \protect\@ordinalstringM{#2}{\@fc@ordstr}%
1367 }%
1368 }%

```

```

1369 \expandafter\edef\csname @fcs@#1\endcsname{%
1370   \noexpand\MakeUppercase{\@fc@ordstr}%
1371 }%
1372 }

```

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

```

1373 \newcommand*\ORDINALstring}[1]{%
1374   \expandafter\protect\expandafter\ORDINALstringnum{%
1375     \expandafter\the\csname c@#1\endcsname
1376   }%
1377 }

```

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

```

1378 \newcommand*\ORDINALstringnum}[1]{%
1379   \new@ifnextchar[%
1380     {\@ORDINAL@string{#1}}%
1381     {\@ORDINAL@string{#1}[m]}%
1382 }

```

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

```

1383 \def\@ORDINAL@string#1[#2]{%
1384   {%
1385     \ifthenelse{\equal{#2}{f}}%
1386     {%
1387       \protect\@ordinalstringF{#1}{\@fc@ordstr}%
1388     }%
1389     {%
1390       \ifthenelse{\equal{#2}{n}}%
1391       {%
1392         \protect\@ordinalstringN{#1}{\@fc@ordstr}%
1393       }%
1394       {%
1395         \ifthenelse{\equal{#2}{m}}%
1396         {}%
1397         {%
1398           \PackageError{fmtcount}%
1399             {Invalid gender option ‘#2’}%
1400             {Available options are m, f or n}%
1401         }%
1402         \protect\@ordinalstringM{#1}{\@fc@ordstr}%
1403       }%
1404     }%
1405     \MakeUppercase{\@fc@ordstr}%
1406   }%
1407 }

```

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

```

1408 \newcommand*\storenumberstring}[2]{%
1409   \expandafter\protect\expandafter\storenumberstringnum{#1}{%
1410     \expandafter\the\csname c@#2\endcsname}%
1411 }

```

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

```

1412 \newcommand*\storenumberstringnum}[2]{%
1413   \@ifnextchar[%
1414     {\@store@number@string{#1}{#2}}%
1415     {\@store@number@string{#1}{#2}[m]}%
1416 }

```

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

```

1417 \def\@store@number@string#1#2[#3]{%
1418   \ifthenelse{\equal{#3}{f}}%
1419     {%
1420       \protect\@numberstringF{#2}{\@fc@numstr}%
1421     }%
1422     {%
1423       \ifthenelse{\equal{#3}{n}}%
1424         {%
1425           \protect\@numberstringN{#2}{\@fc@numstr}%
1426         }%
1427         {%
1428           \ifthenelse{\equal{#3}{m}}%
1429             {}%
1430             {%
1431               \PackageError{fmtcount}
1432                 {Invalid gender option ‘#3’}%
1433                 {Available options are m, f or n}%
1434             }%
1435           \protect\@numberstringM{#2}{\@fc@numstr}%
1436         }%
1437     }%
1438   \expandafter\let\csname @fcs@#1\endcsname\@fc@numstr
1439 }

```

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

```

1440 \newcommand*\numberstring}[1]{%
1441   \expandafter\protect\expandafter\numberstringnum{%
1442     \expandafter\the\csname c@#1\endcsname}%
1443 }

```

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

```

1444 \newcommand*\numberstringnum}[1]{%
1445   \new@ifnextchar[%
1446     {\@number@string{#1}}%
1447     {\@number@string{#1}[m]}%
1448 }

```

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

```
1449 \def\@number@string#1[#2]{%
1450   {%
1451     \ifthenelse{\equal{#2}{f}}%
1452     {%
1453       \protect\@numberstringF{#1}{\@fc@numstr}%
1454     }%
1455     {%
1456       \ifthenelse{\equal{#2}{n}}%
1457       {%
1458         \protect\@numberstringN{#1}{\@fc@numstr}%
1459       }%
1460       {%
1461         \ifthenelse{\equal{#2}{m}}%
1462         {}%
1463         {%
1464           \PackageError{fmtcount}%
1465             {Invalid gender option ‘#2’}%
1466             {Available options are m, f or n}%
1467         }%
1468         \protect\@numberstringM{#1}{\@fc@numstr}%
1469       }%
1470     }%
1471   \@fc@numstr
1472 }%
1473 }
```

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

```
1474 \newcommand*\storeNumberstring[2]{%
1475   \expandafter\protect\expandafter\storeNumberstringnum{#1}{%
1476     \expandafter\the\csname c@#2\endcsname}%
1477 }
```

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

```
1478 \newcommand\storeNumberstringnum[2]{%
1479   \@ifnextchar[%
1480   {\@store@Number@string{#1}{#2}}%
1481   {\@store@Number@string{#1}{#2}[m]}%
1482 }
```

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

```
1483 \def\@store@Number@string#1#2[#3]{%
1484   \ifthenelse{\equal{#3}{f}}%
1485   {%
1486     \protect\@NumberstringF{#2}{\@fc@numstr}%
1487   }%
1488   {%
1489     \ifthenelse{\equal{#3}{n}}%
```

```

1490   {%
1491     \protect\@NumberstringN{#2}{\@fc@numstr}%
1492   }%
1493   {%
1494     \ifthenelse{\equal{#3}{m}}%
1495     {}%
1496     {%
1497       \PackageError{fmtcount}%
1498       {Invalid gender option ‘#3’}%
1499       {Available options are m, f or n}%
1500     }%
1501     \protect\@NumberstringM{#2}{\@fc@numstr}%
1502   }%
1503 }%
1504 \expandafter\let\csname @fcs@#1\endcsname\@fc@numstr
1505 }

```

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

```

1506 \newcommand*{\Numberstring}[1]{%
1507   \expandafter\protect\expandafter\Numberstringnum{%
1508     \expandafter\the\csname c@#1\endcsname}%
1509 }

```

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

```

1510 \newcommand*{\Numberstringnum}[1]{%
1511   \new@ifnextchar[%
1512   {\@Number@string{#1}}%
1513   {\@Number@string{#1}[m]}%
1514 }

```

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

```

1515 \def\@Number@string#1[#2]{%
1516   {%
1517     \ifthenelse{\equal{#2}{f}}%
1518     {%
1519       \protect\@NumberstringF{#1}{\@fc@numstr}%
1520     }%
1521     {%
1522       \ifthenelse{\equal{#2}{n}}%
1523       {%
1524         \protect\@NumberstringN{#1}{\@fc@numstr}%
1525       }%
1526       {%
1527         \ifthenelse{\equal{#2}{m}}%
1528         {}%
1529         {%
1530           \PackageError{fmtcount}%
1531           {Invalid gender option ‘#2’}%
1532           {Available options are m, f or n}%

```

```

1533     }%
1534     \protect\@NumberstringM{#1}{\@fc@numstr}%
1535     }%
1536     }%
1537     \@fc@numstr
1538 }%
1539 }

```

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

```

1540 \newcommand{\storeNUMBERstring}[2]{%
1541   \expandafter\protect\expandafter\storeNUMBERstringnum{#1}{%
1542     \expandafter\the\csname c@#2\endcsname}%
1543 }

```

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

```

1544 \newcommand{\storeNUMBERstringnum}[2]{%
1545   \@ifnextchar[%
1546     {\@store@NUMBER@string{#1}{#2}}%
1547     {\@store@NUMBER@string{#1}{#2}[m]}%
1548 }

```

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

```

1549 \def\@store@NUMBER@string#1#2[#3]{%
1550   \ifthenelse{\equal{#3}{f}}{%
1551     {%
1552       \protect\@numberstringF{#2}{\@fc@numstr}%
1553     }%
1554     {%
1555       \ifthenelse{\equal{#3}{n}}{%
1556         {%
1557           \protect\@numberstringN{#2}{\@fc@numstr}%
1558         }%
1559         {%
1560           \ifthenelse{\equal{#3}{m}}%
1561             {}%
1562             {%
1563               \PackageError{fmtcount}%
1564                 {Invalid gender option ‘#3’}%
1565                 {Available options are m or f}%
1566             }%
1567           \protect\@numberstringM{#2}{\@fc@numstr}%
1568         }%
1569       }%
1570     \expandafter\edef\csname @fcs@#1\endcsname{%
1571       \noexpand\MakeUppercase{\@fc@numstr}%
1572     }%
1573 }

```

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

```
1574 \newcommand*\NUMBERstring[1]{%
1575   \expandafter\protect\expandafter\NUMBERstringnum{%
1576     \expandafter\the\csname c@#1\endcsname}%
1577 }
```

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

```
1578 \newcommand*\NUMBERstringnum[1]{%
1579   \new@ifnextchar[%
1580     {\NUMBER@string{#1}}%
1581     {\NUMBER@string{#1}[m]}%
1582 }
```

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

```
1583 \def\@NUMBER@string#1[#2]{%
1584   {%
1585     \ifthenelse{\equal{#2}{f}}%
1586     {%
1587       \protect\@numberstringF{#1}{\@fc@numstr}%
1588     }%
1589     {%
1590       \ifthenelse{\equal{#2}{n}}%
1591       {%
1592         \protect\@numberstringN{#1}{\@fc@numstr}%
1593       }%
1594       {%
1595         \ifthenelse{\equal{#2}{m}}%
1596         {}%
1597         {%
1598           \PackageError{fmtcount}%
1599             {Invalid gender option ‘#2’}%
1600             {Available options are m, f or n}%
1601         }%
1602         \protect\@numberstringM{#1}{\@fc@numstr}%
1603       }%
1604     }%
1605     \MakeUppercase{\@fc@numstr}%
1606   }%
1607 }
```

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

```
1608 \providecommand*\binary[1]{%
1609   \expandafter\protect\expandafter\@binary{%
1610     \expandafter\the\csname c@#1\endcsname}%
1611 }
```

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

```

1612 \providecommand*\aaalph}[1]{%
1613   \expandafter\protect\expandafter\@aaalph{%
1614     \expandafter\the\csname c@#1\endcsname}%
1615 }

```

`\AAAalph` As before, but upper case.

```

1616 \providecommand*\AAAalph}[1]{%
1617   \expandafter\protect\expandafter\@AAAalph{%
1618     \expandafter\the\csname c@#1\endcsname}%
1619 }

```

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

```

1620 \providecommand*\abalph}[1]{%
1621   \expandafter\protect\expandafter\@abalph{%
1622     \expandafter\the\csname c@#1\endcsname}%
1623 }

```

`\ABAlph` As above, but upper case.

```

1624 \providecommand*\ABAlph}[1]{%
1625   \expandafter\protect\expandafter\@ABAlph{%
1626     \expandafter\the\csname c@#1\endcsname}%
1627 }

```

`\hexadecimal` Hexadecimal:

```

1628 \providecommand*\hexadecimal}[1]{%
1629   \expandafter\protect\expandafter\@hexadecimal{%
1630     \expandafter\the\csname c@#1\endcsname}%
1631 }

```

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

```

1632 \providecommand*\Hexadecimal}[1]{%
1633   \expandafter\protect\expandafter\@Hexadecimal{%
1634     \expandafter\the\csname c@#1\endcsname}%
1635 }

```

`\octal` Octal:

```

1636 \providecommand*\octal}[1]{%
1637   \expandafter\protect\expandafter\@octal{%
1638     \expandafter\the\csname c@#1\endcsname}%
1639 }

```

`\decimal` Decimal:

```

1640 \providecommand*\decimal}[1]{%
1641   \expandafter\protect\expandafter\@decimal{%
1642     \expandafter\the\csname c@#1\endcsname}%
1643 }

```

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.

1644 \def\@setdef@ultfmtcount{%
1645   \ifcsundef{@ordinalMenglish}{\FCloadlang{english}}{}%
1646   \def\@ordinalstringM{\@ordinalstringMenglish}%
1647   \let\@ordinalstringF=\@ordinalstringMenglish
1648   \let\@ordinalstringN=\@ordinalstringMenglish
1649   \def\@OrdinalstringM{\@OrdinalstringMenglish}%
1650   \let\@OrdinalstringF=\@OrdinalstringMenglish
1651   \let\@OrdinalstringN=\@OrdinalstringMenglish
1652   \def\@NumberstringM{\@NumberstringMenglish}%
1653   \let\@NumberstringF=\@NumberstringMenglish
1654   \let\@NumberstringN=\@NumberstringMenglish
1655   \def\@NumberstringM{\@NumberstringMenglish}%
1656   \let\@NumberstringF=\@NumberstringMenglish
1657   \let\@NumberstringN=\@NumberstringMenglish
1658   \def\@ordinalM{\@ordinalMenglish}%
1659   \let\@ordinalF=\@ordinalM
1660   \let\@ordinalN=\@ordinalM
1661 }

\fc@multiling \fc@multiling{<name>}{<gender>}

1662 \newcommand*{\fc@multiling}[2]{%
1663   \ifcsundef{@#1#2\languagename}%
1664   {% try loading it
1665     \FCloadlang{\languagename}%
1666   }%
1667   {%
1668   }%
1669   \ifcsundef{@#1#2\languagename}%
1670   {%
1671     \PackageWarning{fmtcount}%
1672     {No support for \expandafter\protect\csname #1\endcsname\space for
1673     language '\languagename'}%
1674     \ifthenelse{equal{\languagename}{\fc@mainlang}}{%
1675     }%
1676     \FCloadlang{english}%
1677   }%
1678   {%
1679   }%
1680   \ifcsdef{@#1#2\fc@mainlang}%
1681   {%
1682     \csuse{@#1#2\fc@mainlang}%
1683   }%
1684   {%
1685     \PackageWarningNoLine{fmtcount}%

```

```

1686      {No languages loaded at all! Loading english definitions}%
1687      \FCloadlang{english}%
1688      \def\fc@mainlang{english}%
1689      \csuse{@#1#2english}%
1690    }%
1691  }%
1692  {%
1693    \csuse{@#1#2\language}%
1694  }%
1695 }

```

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

```

1696 \def\@set@multiling@fmtcount{%
  The masculine version of \numberstring:
1697   \def\@numberstringM{%
1698     \fc@multiling{numberstring}{M}%
1699   }%
  The feminine version of \numberstring:
1700   \def\@numberstringF{%
1701     \fc@multiling{numberstring}{F}%
1702   }%
  The neuter version of \numberstring:
1703   \def\@numberstringN{%
1704     \fc@multiling{numberstring}{N}%
1705   }%
  The masculine version of \Numberstring:
1706   \def\@NumberstringM{%
1707     \fc@multiling{Numberstring}{M}%
1708   }%
  The feminine version of \Numberstring:
1709   \def\@NumberstringF{%
1710     \fc@multiling{Numberstring}{F}%
1711   }%
  The neuter version of \Numberstring:
1712   \def\@NumberstringN{%
1713     \fc@multiling{Numberstring}{N}%
1714   }%
  The masculine version of \ordinal:
1715   \def\@ordinalM{%
1716     \fc@multiling{ordinal}{M}%
1717   }%
  The feminine version of \ordinal:
1718   \def\@ordinalF{%
1719     \fc@multiling{ordinal}{F}%
1720   }%

```

The neuter version of `\ordinal`:

```
1721 \def\@ordinalN{%
1722   \fc@multiling{ordinal}{N}%
1723 }%
```

The masculine version of `\ordinalstring`:

```
1724 \def\@ordinalstringM{%
1725   \fc@multiling{ordinalstring}{M}%
1726 }%
```

The feminine version of `\ordinalstring`:

```
1727 \def\@ordinalstringF{%
1728   \fc@multiling{ordinalstring}{F}%
1729 }%
```

The neuter version of `\ordinalstring`:

```
1730 \def\@ordinalstringN{%
1731   \fc@multiling{ordinalstring}{N}%
1732 }%
```

The masculine version of `\Ordinalstring`:

```
1733 \def\@OrdinalstringM{%
1734   \fc@multiling{Ordinalstring}{M}%
1735 }%
```

The feminine version of `\Ordinalstring`:

```
1736 \def\@OrdinalstringF{%
1737   \fc@multiling{Ordinalstring}{F}%
1738 }%
```

The neuter version of `\Ordinalstring`:

```
1739 \def\@OrdinalstringN{%
1740   \fc@multiling{Ordinalstring}{N}%
1741 }%
1742 }
```

Check to see if `babel`, `polyglossia` or `ngerman` packages have been loaded, and if yes set `fmtcount` in `multiling`.

```
1743 \expandafter\@ifpackageloaded
1744 \expandafter{\ifxetex polyglossia\else babel\fi}%
1745 {%
1746   \@set@multiling@fmtcount
1747 }%
1748 {%
1749   \@ifpackageloaded{ngerman}%
1750   {%
1751     \FCloadlang{ngerman}%
1752     \@set@multiling@fmtcount
1753   }%
1754   {%
```

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.

```
1755 \iffmtcount@language@option
1756 \set@multiling@fmtcount
```

Some sanity check at the beginning of document may help the end user understand what is wrong:

```
1757 \AtBeginDocument{%
1758 \ifcsundef{languagename}%
1759 {%
1760 \PackageWarning{fmtcount}{%
1761 '\protect{languagename}' is undefined, you should use package babel/polyglossi
1762 language via package option. Reverting to default language.
1763 }%
1764 \@setdef@ultfmtcount
1765 }{%
1766 \@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.

```
1767 \PackageWarning{fmtcount}{%
1768 Setting '\protect{languagename}' to '\fc@mainlang'.\MessageBreak
1769 Reason is that '\protect{languagename}' was '{languagename}',\MessageBreak
1770 but '{languagename}' was not loaded by fmtcount,\MessageBreak
1771 whereas '\fc@mainlang' was the last language loaded by fmtcount ;
1772 }%
1773 \let{languagename}\fc@mainlang
1774 }
1775 }%
1776 }
1777 \else
1778 \@setdef@ultfmtcount
1779 \fi
1780 }%
1781 }
```

Backwards compatibility:

```
1782 \let\@ordinal=\@ordinalM
1783 \let\@ordinalstring=\@ordinalstringM
1784 \let\@Ordinalstring=\@OrdinalstringM
1785 \let\@numberstring=\@numberstringM
1786 \let\@Numberstring=\@NumberstringM
```

9.4.1 fc-american.def

American English definitions

1787 \ProvidesFCLanguage{american}[2013/08/17]%

Loaded fc-USenglish.def if not already loaded

1788 \FCloadlang{USenglish}%

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

1789 \global\let\@ordinalMamerican\@ordinalMUSenglish

1790 \global\let\@ordinalFamerican\@ordinalMUSenglish

1791 \global\let\@ordinalNamerican\@ordinalMUSenglish

1792 \global\let\@numberstringMamerican\@numberstringMUSenglish

1793 \global\let\@numberstringFamerican\@numberstringMUSenglish

1794 \global\let\@numberstringNamerican\@numberstringMUSenglish

1795 \global\let\@NumberstringMamerican\@NumberstringMUSenglish

1796 \global\let\@NumberstringFamerican\@NumberstringMUSenglish

1797 \global\let\@NumberstringNamerican\@NumberstringMUSenglish

1798 \global\let\@ordinalstringMamerican\@ordinalstringMUSenglish

1799 \global\let\@ordinalstringFamerican\@ordinalstringMUSenglish

1800 \global\let\@ordinalstringNamerican\@ordinalstringMUSenglish

1801 \global\let\@OrdinalstringMamerican\@OrdinalstringMUSenglish

1802 \global\let\@OrdinalstringFamerican\@OrdinalstringMUSenglish

1803 \global\let\@OrdinalstringNamerican\@OrdinalstringMUSenglish

9.4.2 fc-british.def

British definitions

1804 \ProvidesFCLanguage{british}[2013/08/17]%

Load fc-english.def, if not already loaded

1805 \FCloadlang{english}%

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

1806 \global\let\@ordinalMbritish\@ordinalMenglish

1807 \global\let\@ordinalFbritish\@ordinalMenglish

1808 \global\let\@ordinalNbritish\@ordinalMenglish

1809 \global\let\@numberstringMbritish\@numberstringMenglish

1810 \global\let\@numberstringFbritish\@numberstringMenglish

1811 \global\let\@numberstringNbritish\@numberstringMenglish

1812 \global\let\@NumberstringMbritish\@NumberstringMenglish

1813 \global\let\@NumberstringFbritish\@NumberstringMenglish

1814 \global\let\@NumberstringNbritish\@NumberstringMenglish

1815 \global\let\@ordinalstringMbritish\@ordinalstringMenglish

1816 \global\let\@ordinalstringFbritish\@ordinalstringMenglish

1817 \global\let\@ordinalstringNbritish\@ordinalstringMenglish

1818 \global\let\@OrdinalstringMbritish\@OrdinalstringMenglish

1819 \global\let\@OrdinalstringFbritish\@OrdinalstringMenglish

1820 \global\let\@OrdinalstringNbritish\@OrdinalstringMenglish

9.4.3 fc-english.def

English definitions

1821 \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.

```

1822 \newcommand*\@ordinalMenglish[2]{%
1823 \def\@fc@ord{}%
1824 \@orgargctr=#1\relax
1825 \@ordinalctr=#1%
1826 \@FCmodulo{\@ordinalctr}{100}%
1827 \ifnum\@ordinalctr=11\relax
1828 \def\@fc@ord{th}%
1829 \else
1830 \ifnum\@ordinalctr=12\relax
1831 \def\@fc@ord{th}%
1832 \else
1833 \ifnum\@ordinalctr=13\relax
1834 \def\@fc@ord{th}%
1835 \else
1836 \@FCmodulo{\@ordinalctr}{10}%
1837 \ifcase\@ordinalctr
1838 \def\@fc@ord{th}% case 0
1839 \or \def\@fc@ord{st}% case 1
1840 \or \def\@fc@ord{nd}% case 2
1841 \or \def\@fc@ord{rd}% case 3
1842 \else
1843 \def\@fc@ord{th}% default case
1844 \fi
1845 \fi
1846 \fi
1847 \fi
1848 \edef#2{\number#1\relax\noexpand\fmtord{\@fc@ord}}%
1849 }%
1850 \global\let\@ordinalMenglish\@ordinalMenglish

```

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

```

1851 \global\let\@ordinalFenglish=\@ordinalMenglish
1852 \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.

```

1853 \newcommand*\@unitstringenglish[1]{%
1854 \ifcase#1\relax
1855 zero%
1856 \or one%
1857 \or two%
1858 \or three%
1859 \or four%
1860 \or five%

```

```

1861 \or six%
1862 \or seven%
1863 \or eight%
1864 \or nine%
1865 \fi
1866 }%
1867 \global\let\@@unitstringenglish\@@unitstringenglish

```

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

```

1868 \newcommand*\@@tenstringenglish[1]{%
1869 \ifcase#1\relax
1870 \or ten%
1871 \or twenty%
1872 \or thirty%
1873 \or forty%
1874 \or fifty%
1875 \or sixty%
1876 \or seventy%
1877 \or eighty%
1878 \or ninety%
1879 \fi
1880 }%
1881 \global\let\@@tenstringenglish\@@tenstringenglish

```

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

```

1882 \newcommand*\@@teenstringenglish[1]{%
1883 \ifcase#1\relax
1884 ten%
1885 \or eleven%
1886 \or twelve%
1887 \or thirteen%
1888 \or fourteen%
1889 \or fifteen%
1890 \or sixteen%
1891 \or seventeen%
1892 \or eighteen%
1893 \or nineteen%
1894 \fi
1895 }%
1896 \global\let\@@teenstringenglish\@@teenstringenglish

```

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

```

1897 \newcommand*\@@Unitstringenglish[1]{%
1898 \ifcase#1\relax
1899 Zero%
1900 \or One%
1901 \or Two%
1902 \or Three%
1903 \or Four%
1904 \or Five%
1905 \or Six%

```

```

1906 \or Seven%
1907 \or Eight%
1908 \or Nine%
1909 \fi
1910 }%
1911 \global\let\@@Unitstringenglish\@@Unitstringenglish

```

The tens:

```

1912 \newcommand*\@@Tenstringenglish[1]{%
1913 \ifcase#1\relax
1914 \or Ten%
1915 \or Twenty%
1916 \or Thirty%
1917 \or Forty%
1918 \or Fifty%
1919 \or Sixty%
1920 \or Seventy%
1921 \or Eighty%
1922 \or Ninety%
1923 \fi
1924 }%
1925 \global\let\@@Tenstringenglish\@@Tenstringenglish

```

The teens:

```

1926 \newcommand*\@@Teenstringenglish[1]{%
1927 \ifcase#1\relax
1928 Ten%
1929 \or Eleven%
1930 \or Twelve%
1931 \or Thirteen%
1932 \or Fourteen%
1933 \or Fifteen%
1934 \or Sixteen%
1935 \or Seventeen%
1936 \or Eighteen%
1937 \or Nineteen%
1938 \fi
1939 }%
1940 \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.)

```

1941 \newcommand*\@@numberstringenglish[2]{%
1942 \ifnum#1>99999
1943 \PackageError{fmtcount}{Out of range}%
1944 {This macro only works for values less than 100000}%
1945 \else
1946 \ifnum#1<0

```

```

1947 \PackageError{fmtcount}{Negative numbers not permitted}%
1948 {This macro does not work for negative numbers, however
1949 you can try typing "minus" first, and then pass the modulus of
1950 this number}%
1951 \fi
1952 \fi
1953 \def#2{}%
1954 \@strctr=#1\relax \divide\@strctr by 1000\relax
1955 \ifnum\@strctr>9
1956   \divide\@strctr by 10
1957   \ifnum\@strctr>1\relax
1958     \let\@fc@numstr#2\relax
1959     \edef#2{\@fc@numstr\@tenstring{\@strctr}}%
1960     \@strctr=#1 \divide\@strctr by 1000\relax
1961     \@FCmodulo{\@strctr}{10}%
1962     \ifnum\@strctr>0\relax
1963       \let\@fc@numstr#2\relax
1964       \edef#2{\@fc@numstr-\@unitstring{\@strctr}}%
1965     \fi
1966   \else
1967     \@strctr=#1\relax
1968     \divide\@strctr by 1000\relax
1969     \@FCmodulo{\@strctr}{10}%
1970     \let\@fc@numstr#2\relax
1971     \edef#2{\@fc@numstr\@teenstring{\@strctr}}%
1972   \fi
1973   \let\@fc@numstr#2\relax
1974   \edef#2{\@fc@numstr\ \@thousand}%
1975 \else
1976   \ifnum\@strctr>0\relax
1977     \let\@fc@numstr#2\relax
1978     \edef#2{\@fc@numstr\@unitstring{\@strctr}\ \@thousand}%
1979   \fi
1980 \fi
1981 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
1982 \divide\@strctr by 100
1983 \ifnum\@strctr>0\relax
1984   \ifnum#1>1000\relax
1985     \let\@fc@numstr#2\relax
1986     \edef#2{\@fc@numstr\ }%
1987   \fi
1988   \let\@fc@numstr#2\relax
1989   \edef#2{\@fc@numstr\@unitstring{\@strctr}\ \@hundred}%
1990 \fi
1991 \@strctr=#1\relax \@FCmodulo{\@strctr}{100}%
1992 \ifnum#1>100\relax
1993   \ifnum\@strctr>0\relax
1994     \let\@fc@numstr#2\relax
1995     \edef#2{\@fc@numstr\ \@andname\ }%

```

```

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

```

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

```

2026 \DeclareRobustCommand{\@numberstringMenglish}[2]{%
2027 \let \@unitstring=\@unitstringenglish
2028 \let \@teenstring=\@teenstringenglish
2029 \let \@tenstring=\@tenstringenglish
2030 \def \@hundred{hundred}\def \@thousand{thousand}%
2031 \def \@andname{and}%
2032 \@numberstringenglish{#1}{#2}%
2033 }%
2034 \global\let \@numberstringMenglish \@numberstringMenglish

```

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

```

2035 \global\let \@numberstringFenglish=\@numberstringMenglish
2036 \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.

```

2037 \newcommand* \@NumberstringMenglish[2]{%

```

```

2038 \let\@unitstring=\@Unitstringenglish
2039 \let\@teenstring=\@Teenstringenglish
2040 \let\@tenstring=\@Tenstringenglish
2041 \def\@hundred{Hundred}\def\@thousand{Thousand}%
2042 \def\@andname{and}%
2043 \@@numberstringenglish{#1}{#2}%
2044 }%
2045 \global\let\@NumberstringMenglish\@NumberstringMenglish

```

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

```

2046 \global\let\@NumberstringFenglish=\@NumberstringMenglish
2047 \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:

```

2048 \newcommand*\@@unitthstringenglish[1]{%
2049   \ifcase#1\relax
2050     zeroth%
2051     \or first%
2052     \or second%
2053     \or third%
2054     \or fourth%
2055     \or fifth%
2056     \or sixth%
2057     \or seventh%
2058     \or eighth%
2059     \or ninth%
2060   \fi
2061 }%
2062 \global\let\@@unitthstringenglish\@@unitthstringenglish

```

Next the tens:

```

2063 \newcommand*\@@tenthstringenglish[1]{%
2064   \ifcase#1\relax
2065     \or tenth%
2066     \or twentieth%
2067     \or thirtieth%
2068     \or fortieth%
2069     \or fiftieth%
2070     \or sixtieth%
2071     \or seventieth%
2072     \or eightieth%
2073     \or ninetieth%
2074   \fi
2075 }%
2076 \global\let\@@tenthstringenglish\@@tenthstringenglish

```

The teens:

```

2077 \newcommand*\@@teenthstringenglish[1]{%
2078   \ifcase#1\relax

```

```

2079 tenth%
2080 \or eleventh%
2081 \or twelfth%
2082 \or thirteenth%
2083 \or fourteenth%
2084 \or fifteenth%
2085 \or sixteenth%
2086 \or seventeenth%
2087 \or eighteenth%
2088 \or nineteenth%
2089 \fi
2090 }%
2091 \global\let\@@teenthstringenglish\@@teenthstringenglish

```

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

```

2092 \newcommand*\@@Unitthstringenglish[1]{%
2093 \ifcase#1\relax
2094 Zeroth%
2095 \or First%
2096 \or Second%
2097 \or Third%
2098 \or Fourth%
2099 \or Fifth%
2100 \or Sixth%
2101 \or Seventh%
2102 \or Eighth%
2103 \or Ninth%
2104 \fi
2105 }%
2106 \global\let\@@Unitthstringenglish\@@Unitthstringenglish

```

The tens:

```

2107 \newcommand*\@@Tenthstringenglish[1]{%
2108 \ifcase#1\relax
2109 \or Tenth%
2110 \or Twentieth%
2111 \or Thirtieth%
2112 \or Fortieth%
2113 \or Fiftieth%
2114 \or Sixtieth%
2115 \or Seventieth%
2116 \or Eightieth%
2117 \or Ninetieth%
2118 \fi
2119 }%
2120 \global\let\@@Tenthstringenglish\@@Tenthstringenglish

```

The teens:

```

2121 \newcommand*\@@Teenthstringenglish[1]{%
2122 \ifcase#1\relax
2123 Tenth%

```

```

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

2136 \newcommand*\@@ordinalstringenglish[2]{%
2137 \@strctr=#1\relax
2138 \ifnum#1>99999
2139 \PackageError{fmtcount}{Out of range}%
2140 {This macro only works for values less than 100000 (value given: \number\@strctr)}%
2141 \else
2142 \ifnum#1<0
2143 \PackageError{fmtcount}{Negative numbers not permitted}%
2144 {This macro does not work for negative numbers, however
2145 you can try typing "minus" first, and then pass the modulus of
2146 this number}%
2147 \fi
2148 \def#2{}%
2149 \fi
2150 \@strctr=#1\relax \divide\@strctr by 1000\relax
2151 \ifnum\@strctr>9\relax

#1 is greater or equal to 10000

2152 \divide\@strctr by 10
2153 \ifnum\@strctr>1\relax
2154 \let\@@fc@ordstr#2\relax
2155 \edef#2{\@@fc@ordstr\@teenstring{\@strctr}}%
2156 \@strctr=#1\relax
2157 \divide\@strctr by 1000\relax
2158 \@FCmodulo{\@strctr}{10}%
2159 \ifnum\@strctr>0\relax
2160 \let\@@fc@ordstr#2\relax
2161 \edef#2{\@@fc@ordstr-\@unitstring{\@strctr}}%
2162 \fi
2163 \else
2164 \@strctr=#1\relax \divide\@strctr by 1000\relax
2165 \@FCmodulo{\@strctr}{10}%
2166 \let\@@fc@ordstr#2\relax
2167 \edef#2{\@@fc@ordstr\@teenstring{\@strctr}}%

```

```

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

```

```

2217 \@FCmodulo{\@tmpstrctr}{10}%
2218 \let\@fc@ordstr#2\relax
2219 \ifnum\@tmpstrctr=0\relax
2220   \edef#2{\@fc@ordstr\@tenthstring{\@strctr}}%
2221 \else
2222   \edef#2{\@fc@ordstr\@tenstring{\@strctr}}%
2223 \fi
2224 \@strctr=#1\relax \@FCmodulo{\@strctr}{10}%
2225 \ifnum\@strctr>0\relax
2226   \let\@fc@ordstr#2\relax
2227   \edef#2{\@fc@ordstr-\@unitthstring{\@strctr}}%
2228 \fi
2229 \else
2230   \ifnum\@strctr<10\relax
2231     \ifnum\@strctr=0\relax
2232       \ifnum#1<100\relax
2233         \let\@fc@ordstr#2\relax
2234         \edef#2{\@fc@ordstr\@unitthstring{\@strctr}}%
2235       \fi
2236     \else
2237       \let\@fc@ordstr#2\relax
2238       \edef#2{\@fc@ordstr\@unitthstring{\@strctr}}%
2239     \fi
2240   \else
2241     \@FCmodulo{\@strctr}{10}%
2242     \let\@fc@ordstr#2\relax
2243     \edef#2{\@fc@ordstr\@teenthstring{\@strctr}}%
2244   \fi
2245 \fi
2246 }%
2247 \global\let\@ordinalstringenglish\@ordinalstringenglish

```

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

```

2248 \DeclareRobustCommand{\@ordinalstringMenglish}[2]{%
2249   \let\@unitthstring=\@unitthstringenglish
2250   \let\@teenthstring=\@teenthstringenglish
2251   \let\@tenthstring=\@tenthstringenglish
2252   \let\@unitstring=\@unitstringenglish
2253   \let\@teenstring=\@teenstringenglish
2254   \let\@tenstring=\@tenstringenglish
2255   \def\@andname{and}%
2256   \def\@hundred{hundred}\def\@thousand{thousand}%
2257   \def\@hundredth{hundredth}\def\@thousandth{thousandth}%
2258   \@ordinalstringenglish{#1}{#2}%
2259 }%
2260 \global\let\@ordinalstringMenglish\@ordinalstringMenglish

```

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

```

2261 \global\let\@ordinalstringFenglish=\@ordinalstringMenglish

```

```
2262 \global\let\@ordinalstringNenglish=\@ordinalstringMenglish
```

First letter of each word in upper case:

```
2263 \DeclareRobustCommand{\@OrdinalstringMenglish}[2]{%
2264   \let\@unitthstring=\@Unitthstringenglish
2265   \let\@teenthstring=\@Teenthstringenglish
2266   \let\@tenthstring=\@Tenthstringenglish
2267   \let\@unitstring=\@Unitstringenglish
2268   \let\@teenstring=\@Teenstringenglish
2269   \let\@tenstring=\@Tenstringenglish
2270   \def\@andname{and}%
2271   \def\@hundred{Hundred}\def\@thousand{Thousand}%
2272   \def\@hundredth{Hundredth}\def\@thousandth{Thousandth}%
2273   \@ordinalstringenglish{#1}{#2}%
2274 }%
2275 \global\let\@OrdinalstringMenglish\@OrdinalstringMenglish
```

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

```
2276 \global\let\@OrdinalstringFenglish=\@OrdinalstringMenglish
2277 \global\let\@OrdinalstringNenglish=\@OrdinalstringMenglish
```

9.4.4 fc-francais.def

```
2278 \ProvidesFCLanguage{francais}[2013/08/17]%
2279 \FCLoadlang{french}%
```

Set francais to be equivalent to french.

```
2280 \global\let\@ordinalMfrancais=\@ordinalMfrench
2281 \global\let\@ordinalFfrancais=\@ordinalFfrench
2282 \global\let\@ordinalNfrancais=\@ordinalNfrench
2283 \global\let\@numberstringMfrancais=\@numberstringMfrench
2284 \global\let\@numberstringFfrancais=\@numberstringFfrench
2285 \global\let\@numberstringNfrancais=\@numberstringNfrench
2286 \global\let\@NumberstringMfrancais=\@NumberstringMfrench
2287 \global\let\@NumberstringFfrancais=\@NumberstringFfrench
2288 \global\let\@NumberstringNfrancais=\@NumberstringNfrench
2289 \global\let\@ordinalstringMfrancais=\@ordinalstringMfrench
2290 \global\let\@ordinalstringFfrancais=\@ordinalstringFfrench
2291 \global\let\@ordinalstringNfrancais=\@ordinalstringNfrench
2292 \global\let\@OrdinalstringMfrancais=\@OrdinalstringMfrench
2293 \global\let\@OrdinalstringFfrancais=\@OrdinalstringFfrench
2294 \global\let\@OrdinalstringNfrancais=\@OrdinalstringNfrench
```

9.4.5 fc-french.def

Definitions for French.

```
2295 \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'.
2296 \def\fc@french@set@plural#1#2#3#4#5#6{%
2297   \ifthenelse{\equal{#2}{reformed}}{%
2298     \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{#3}%
2299   }{%
2300     \ifthenelse{\equal{#2}{traditional}}{%
2301       \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{#4}%
2302     }{%
2303       \ifthenelse{\equal{#2}{reformed o}}{%
2304         \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{#5}%
2305       }{%
2306         \ifthenelse{\equal{#2}{traditional o}}{%
2307           \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{#6}%
2308         }{%
2309           \ifthenelse{\equal{#2}{always}}{%
2310             \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{0}%
2311           }{%
2312             \ifthenelse{\equal{#2}{never}}{%
2313               \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{1}%
2314             }{%
2315               \ifthenelse{\equal{#2}{multiple}}{%
2316                 \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{2}%
2317               }{%
2318                 \ifthenelse{\equal{#2}{multiple g-last}}{%
2319                   \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{3}%
2320                 }{%
2321                   \ifthenelse{\equal{#2}{multiple l-last}}{%
2322                     \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{4}%
2323                   }{%
2324                     \ifthenelse{\equal{#2}{multiple lng-last}}{%
2325                       \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{5}%
2326                     }{%
2327                       \ifthenelse{\equal{#2}{multiple ng-last}}{%
2328                         \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{6}%
2329                       }{%
2330                         \PackageError{fmtcount}{Unexpected argument}{%
2331                           '#2' was unexpected: french option '#1 plural' expects 'reformed'
2332                           'reformed o', 'traditional o', 'always', 'never', 'multiple', 'mu
2333                           'multiple l-last', 'multiple lng-last', or 'multiple ng-last'.%
2334                         }}}}

```

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’.

```

2335 \def\@tempa#1#2#3{%
2336   \define@key{fcfrench}{#1 plural}[reformed]{%
2337     \fc@french@set@plural{#1}{##1}{#2}{#3}{#3}%
2338   }%
2339 }
2340 \@tempa{vingt}{4}{5}
2341 \@tempa{cent}{4}{5}
2342 \@tempa{mil}{0}{0}
2343 \@tempa{n-illion}{2}{6}
2344 \@tempa{n-illiard}{2}{6}

```

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

```

2345 \define@key{fcfrench}{all plural}[reformed]{%
2346   \csname KV@fcfrench@vingt plural\endcsname{#1}%
2347   \csname KV@fcfrench@cent plural\endcsname{#1}%
2348   \csname KV@fcfrench@mil plural\endcsname{#1}%
2349   \csname KV@fcfrench@n-illion plural\endcsname{#1}%
2350   \csname KV@fcfrench@n-illiard plural\endcsname{#1}%
2351 }

```

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. <n>illion and <n>illiard,
always	always use dashes to separate all words

```

2352 \define@key{fcfrench}{dash or space}[reformed]{%
2353   \ifthenelse{\equal{#1}{traditional}}{%
2354     \let\fc@frenchoptions@supermillion@dos\space%
2355     \let\fc@frenchoptions@submillion@dos\space
2356   }{%
2357     \ifthenelse{\equal{#1}{reformed}\or\equal{#1}{1990}}{%
2358       \let\fc@frenchoptions@supermillion@dos\space
2359       \def\fc@frenchoptions@submillion@dos{-}%
2360     }{%
2361       \ifthenelse{\equal{#1}{always}}{%
2362         \def\fc@frenchoptions@supermillion@dos{-}%
2363         \def\fc@frenchoptions@submillion@dos{-}%
2364       }{%
2365         \PackageError{fmtcount}{Unexpected argument}{%
2366           French option ‘dash or space’ expects ‘always’, ‘reformed’ or ‘traditional’
2367         }
2368       }%
2369     }%

```

```

2370 }%
2371 }
    Option ‘scale’, can take 3 possible values:
        long   for which  $\langle n \rangle$ illions &  $\langle n \rangle$ illiards are used with  $10^{6 \times n} = 1 \langle n \rangle$ illion, and  $10^{6 \times n + 3} = 1 \langle n \rangle$ illiard
        short  for which  $\langle n \rangle$ illions only are used with  $10^{3 \times n + 3} = 1 \langle n \rangle$ illion
        recursive for which  $10^{18} = \text{un milliard de milliards}$ 
2372 \define@key{fcfrench}{scale}[recursive]{%
2373   \ifthenelse{\equal{#1}{long}}{%
2374     \let\fc@poweroften\fc@pot@longscalefrench
2375   }{%
2376     \ifthenelse{\equal{#1}{recursive}}{%
2377       \let\fc@poweroften\fc@pot@recursivefrench
2378     }{%
2379       \ifthenelse{\equal{#1}{short}}{%
2380         \let\fc@poweroften\fc@pot@shortscalefrench
2381       }{%
2382         \PackageError{fmtcount}{Unexpected argument}{%
2383           French option ‘scale’ expects ‘long’, ‘recursive’ or ‘short’
2384         }
2385       }%
2386     }%
2387   }%
2388 }

```

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”

```

2389 \define@key{fcfrench}{n-illiard upto}[infinity]{%
2390   \ifthenelse{\equal{#1}{infinity}}{%
2391     \def\fc@longscale@nilliard@upto{0}%
2392   }{%
2393     \ifthenelse{\equal{#1}{infty}}{%
2394       \def\fc@longscale@nilliard@upto{0}%
2395     }{%
2396       \if Q\ifnum9<1#1Q\fi\else
2397         \PackageError{fmtcount}{Unexpected argument}{%
2398           French option ‘milliard threshold’ expects ‘infinity’, or equivalently ‘infty’, or
2399           integer.}%
2400       \fi
2401       \def\fc@longscale@nilliard@upto{#1}%
2402     }%
2403 }

```

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.

```
2404 \def\@tempa#1{%
2405   \define@key{fcfrench}{#1}[]{%
2406     \PackageError{fmtcount}{Unexpected argument}{French option with key ‘#1’ does not take
2407       any value}}%
2408   \expandafter\def\csname KV@fcfrench@#1@default\endcsname{%
2409     \def\fmtcount@french{#1}}%
2410 }%
2411 \@tempa{france}\@tempa{swiss}\@tempa{belgian}%
```

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

```
2412 \define@key{fcfrench}{dialect}[france]{%
2413   \ifthenelse{\equal{#1}{france}
2414     \or\equal{#1}{swiss}
2415     \or\equal{#1}{belgian}}{%
2416     \def\fmtcount@french{#1}}{%
2417     \PackageError{fmtcount}{Invalid value ‘#1’ to french option dialect key}
2418     {Option ‘french’ can only take the values ‘france’,
2419       ‘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’.

```
2420 \define@key{fcfrench}{mil plural mark}[le]{%
2421   \def\fc@frenchoptions@mil@plural@mark{#1}}
```

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

```
2422 \def\fc@UpperCaseFirstLetter#1#2\@nil{%
2423   \uppercase{#1}#2}
2424
2425 \def\fc@CaseIden#1\@nil{%
2426   #1%
2427 }
2428 \def\fc@UpperCaseAll#1\@nil{%
2429   \uppercase{#1}%
2430 }
2431
2432 \let\fc@case\fc@CaseIden
2433
```

\@ ordinalMfrench

```
2434 \newcommand*{\@ordinalMfrench}[2]{%
2435   \iffmtord@abbrv
2436   \edef#2{\number#1\relax\noexpand\fmtord{e}}%
2437 \else
2438   \ifnum#1=1\relax
2439     \edef#2{\number#1\relax\noexpand\fmtord{er}}%
2440 \else
```

```

2441 \edef#2{\number#1\relax\noexpand\fmtord{eme}}%
2442 \fi
2443 \fi}

```

\@ ordinalFfrench

```

2444 \newcommand*{\@ordinalFfrench}[2]{%
2445 \iffmtord@abbrv
2446 \edef#2{\number#1\relax\noexpand\fmtord{e}}%
2447 \else
2448 \ifnum#1=1 %
2449 \edef#2{\number#1\relax\noexpand\fmtord{i\`ere}}%
2450 \else
2451 \edef#2{\number#1\relax\noexpand\fmtord{i\`eme}}%
2452 \fi
2453 \fi}

```

In French neutral gender and masculine gender are formally identical.

```

2454 \let\@ordinalNfrench\@ordinalMfrench

```

\@ @unitstringfrench

```

2455 \newcommand*{\@@unitstringfrench}[1]{%
2456 \noexpand\fc@case
2457 \ifcase#1 %
2458 z\`ero%
2459 \or un%
2460 \or deux%
2461 \or trois%
2462 \or quatre%
2463 \or cinq%
2464 \or six%
2465 \or sept%
2466 \or huit%
2467 \or neuf%
2468 \fi
2469 \noexpand\@nil
2470 }

```

\@ @tenstringfrench

```

2471 \newcommand*{\@@tenstringfrench}[1]{%
2472 \noexpand\fc@case
2473 \ifcase#1 %
2474 \or dix%
2475 \or vingt%
2476 \or trente%
2477 \or quarante%
2478 \or cinquante%
2479 \or soixante%
2480 \or septante%
2481 \or huitante%
2482 \or nonante%
2483 \or cent%

```

```

2484 \fi
2485 \noexpand\@nil
2486 }

\@ @teenstringfrench
2487 \newcommand*\@@teenstringfrench}[1]{%
2488 \noexpand\fc@case
2489 \ifcase#1 %
2490     dix%
2491 \or onze%
2492 \or douze%
2493 \or treize%
2494 \or quatorze%
2495 \or quinze%
2496 \or seize%
2497 \or dix\noexpand\@nil-\noexpand\fc@case sept%
2498 \or dix\noexpand\@nil-\noexpand\fc@case huit%
2499 \or dix\noexpand\@nil-\noexpand\fc@case neuf%
2500 \fi
2501 \noexpand\@nil
2502 }

\@ @seventiesfrench
2503 \newcommand*\@@seventiesfrench}[1]{%
2504 \@tenstring{6}%
2505 \ifnum#1=1 %
2506 \fc@frenchoptions@submillion@dos\@andname\fc@frenchoptions@submillion@dos
2507 \else
2508 -%
2509 \fi
2510 \@teenstring{#1}%
2511 }

\@ @eightiesfrench Macro \@@eightiesfrench is used to format numbers in the
interval [80..89]. Argument as follows:
#1 digit  $d_w$  such that the number to be formatted is  $80 + d_w$ 
Implicit arguments as:
\count0 weight  $w$  of the number  $d_{w+1}d_w$  to be formatted
\count1 same as \#1
\count6 input, counter giving the least weight of non zero digits in top level
formatted number integral part, with rounding down to a multiple
of 3,
\count9 input, counter giving the power type of the power of ten follow-
ing the eighties to be formatted; that is '1' for "mil" and '2' for
"(n)illion|(n)illiard".
2512 \newcommand*\@@eightiesfrench[1]{%
2513 \fc@case quatre\@nil-\noexpand\fc@case vingt%
2514 \ifnum#1>0 %
2515 \ifnum\fc@frenchoptions@vingt@plural=0 % vingt plural=always

```

```

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

```

```

2565 \ifnum#1>0 \@unitstring{#1}\fi
2566 }
2567 \newcommand*{\@@eightiesfrenchswiss}[1]{%
2568 \@tenstring{8}%
2569 \ifnum#1=1\ \@andname\ \fi
2570 \ifnum#1>1-\fi
2571 \ifnum#1>0 \@unitstring{#1}\fi
2572 }
2573 \newcommand*{\@@ninetiesfrenchswiss}[1]{%
2574 \@tenstring{9}%
2575 \ifnum#1=1\ \@andname\ \fi
2576 \ifnum#1>1-\fi
2577 \ifnum#1>0 \@unitstring{#1}\fi
2578 }

```

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

```

2579 \newcommand*\fc@french@common{%
2580 \let\@unitstring=\@unitstringfrench
2581 \let\@teenstring=\@teenstringfrench
2582 \let\@tenstring=\@tenstringfrench
2583 \def\@hundred{cent}%
2584 \def\@andname{et}%
2585 }

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

2807 \fc@french@common
2808 \let\@seventies=\@seventiesfrench
2809 \let\@eighties=\@eightiesfrench
2810 \let\@nineties=\@ninetiesfrench
2811 \@@ordinalstringfrench{#1}{#2}%
2812 }
2813 \let\@OrdinalstringFfrench=\@OrdinalstringFfrenchfrance
2814 \let\@OrdinalstringNfrench\@OrdinalstringMfrench

```

`\fc @@@@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.

```

2815 \ifcsundef{fc@@do@plural@mark}{}%
2816 {\PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
2817   '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.

```

2818 \def\fc@@do@plural@mark#1{%
2819   \ifcase\count10 %
2820     #1% 0=always
2821     \or% 1=never
2822     \or% 2=multiple
2823     \ifnum\count1>1 %
2824       #1%
2825       \fi
2826     \or% 3= multiple g-last
2827     \ifnum\count1>1 %
2828       \ifnum\count0=\count6 %
2829         #1%
2830         \fi
2831       \fi
2832     \or% 4= multiple l-last
2833     \ifnum\count1>1 %
2834       \ifnum\count9=1 %
2835       \else

```

```

2836         #1%
2837         \fi
2838     \fi
2839 \or% 5= multiple lng-last
2840     \ifnum\count1>1 %
2841         \ifnum\count9=1 %
2842     \else
2843         \if\count0>\count6 %
2844             #1%
2845         \fi
2846     \fi
2847 \fi
2848 \or% 6= multiple ng-last
2849     \ifnum\count1>1 %
2850         \ifnum\count0>\count6 %
2851             #1%
2852     \fi
2853 \fi
2854 \fi
2855 }

```

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

```

2856 \ifcsundef{fc@@nbrstr@Fpreamble}{}{%
2857     \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
2858         'fc@@nbrstr@Fpreamble'}}

```

\fc @@nbrstr@Fpreamble

```

2859 \def\fc@@nbrstr@Fpreamble{%
2860     \fc@read@unit{\count1}{0}%
2861     \ifnum\count1=1 %
2862         \let\fc@case@save\fc@case
2863         \def\fc@case{\noexpand\fc@case}%
2864         \def\@nil{\noexpand\@nil}%
2865         \let\fc@nbrstr@postamble\fc@@nbrstr@Fpostamble
2866     \fi
2867 }

```

\fc @@nbrstr@Fpostamble

```

2868 \def\fc@@nbrstr@Fpostamble{%
2869     \let\fc@case\fc@case@save
2870     \expandafter\fc@get@last@word\expandafter{\@tempa}\@tempb\@tempc
2871     \def\@tempd{un}%
2872     \ifx\@tempc\@tempd
2873         \let\@tempc\@tempa
2874         \edef\@tempa{\@tempb\fc@case une\@nil}%
2875     \fi
2876 }

```

\fc @@pot@longscalefrench Macro \fc@@pot@longscalefrench is used to produce powers of ten with long scale convention. The long scale convention is

correct for French and elsewhere in Europe. First we check that the macro is not yet defined.

```
2877 \ifcsundef{fc@pot@longscalefrench}{}{%
2878   \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
2879     '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

```
2880 \def\fc@pot@longscalefrench#1#2#3{%
2881   {%
```

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

```
2882   \edef\@tempb{\number#1}%
```

Let `\count1` be the plural value.

```
2883   \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 .

```
2884   \count2\count0 %
2885   \divide\count2 by 6 %
2886   \count3\count2 %
2887   \multiply\count3 by 6 %
2888   \count3-\count3 %
2889   \advance\count3 by \count0 %
2890   \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)”.

```
2891   \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.

```
2892   \edef\@temp{%
2893     \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.

```
2894     1%
```

```

2895         \else
2896         \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`.

```

2897         \ifnum\fc@longscale@nilliard@upto=0 %

```

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

```

2898             2%
2899         \else

```

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

```

2900             \ifnum\count2>\fc@longscale@nilliard@upto
2901             1%
2902         \else
2903             2%
2904         \fi
2905     \fi
2906 \else
2907     2%
2908 \fi
2909 \fi
2910 }%
2911 \ifnum\@temph=1 %

```

Here 10^w is formatted as “mil(le)”.

```

2912     \count10=\fc@frenchoptions@mil@plural\space
2913     \edef\@tempe{%
2914         \noexpand\fc@case
2915         mil%
2916         \fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark
2917         \noexpand\@nil
2918     }%
2919 \else
2920     % weight >= 6
2921     \expandafter\fc@@latin@cardinal@pefix\expandafter{\the\count2}\@tempg
2922     % now form the xxx-illion(s) or xxx-illiard(s) word
2923     \ifnum\count3>2 %
2924         \toks10{illiard}%
2925         \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space
2926     \else
2927         \toks10{illion}%
2928         \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
2929     \fi
2930     \edef\@tempe{%
2931         \noexpand\fc@case
2932         \@tempg
2933         \the\toks10 %
2934         \fc@@do@plural@mark s%

```

```

2935         \noexpand\@nil
2936     }%
2937     \fi
2938     \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.

```

2939         \let\@tempe\@empty
2940     \def\@temph{0}%
2941     \fi
2942     \else

```

Case of $w = 0$.

```

2943         \let\@tempe\@empty
2944     \def\@temph{0}%
2945     \fi

```

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

```

2946     \expandafter\toks\expandafter1\expandafter{\@tempe}%
2947     \toks0{#2}%
2948     \edef\@tempa{\the\toks0 \@temph \def\noexpand#3{\the\toks1}}%
2949     \expandafter
2950 } \@tempa
2951 }

```

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

```

2952 \ifcsundef{fc@@pot@shortscalefrench}{-}{%
2953     \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
2954     '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

```

2955 \def\fc@@pot@shortscalefrench#1#2#3{%
2956     {%

```

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

```

2957     \edef\@tempb{\number#1}%

```

And let `\count1` be the plural value.

```
2958 \count1=\@tempb
```

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

```
2959 \count2\count0 %
2960 \divide\count2 by 3 %
2961 \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 `\@temp`. 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)”.

```
2962 \ifnum\count0>0 % If weight>=3, i.e we do have to append thousand or n-illion(s)/n-illi.
2963 \ifnum\count1>0 % we have at least one thousand/n-illion/n-illiard
2964 \ifnum\count2=0 %
2965 \def\@temp{1}%
2966 \count1=\fc@frenchoptions@mil@plural\space
2967 \edef\@tempe{%
2968 mil%
2969 \fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark
2970 }%
2971 \else
2972 \def\@temp{2}%
2973 % weight >= 6
2974 \expandafter\fc@latin@cardinal@prefix\expandafter{\the\count2}\@tempg
2975 \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
2976 \edef\@tempe{%
2977 \noexpand\fc@case
2978 \@tempg
2979 illion%
2980 \fc@@do@plural@mark s%
2981 \noexpand\@nil
2982 }%
2983 \fi
2984 \else
```

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

```
2985 \def\@temp{0}%
2986 \let\@tempe\@empty
2987 \fi
2988 \else
```

Here $w = 0$.

```
2989 \def\@temp{0}%
2990 \let\@tempe\@empty
2991 \fi
2992 % now place into \cs{@tempa} the assignment of results \cs{@temp} and \cs{@tempe} to to \
2993 % \texttt{\#3} for further propagation after closing brace.
2994 % \begin{macrocode}
```

```

2995 \expandafter\toks\expandafter1\expandafter{\@tempe}%
2996 \toks0{#2}%
2997 \edef\@tempa{\the\toks0 \@tempb \def\noexpand#3{\the\toks1}}%
2998 \expandafter
2999 }\@tempa
3000 }

```

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

```

3001 \ifcsundef{fc@@pot@recursivefrench}{-}{%
3002 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
3003 '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

```

3004 \def\fc@@pot@recursivefrench#1#2#3{%
3005  {%

```

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

```

3006 \edef\@tempb{\number#1}%
3007 \let\@tempa\@tempa

```

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

```

3008 \count1=\@tempb\space

```

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

```

3009 \ifnum\count1>0 %
3010 \count2\count0 %
3011 \divide\count2 by 9 %
3012 \advance\count2 by -1 %
3013 \let\@tempe\@empty
3014 \edef\@tempf{\fc@frenchoptions@supermillion@dos
3015 de\fc@frenchoptions@supermillion@dos\fc@case milliards\@nil}%
3016 \count11\count0 %
3017 \ifnum\count2>0 %
3018 \count3\count2 %
3019 \count3-\count3 %
3020 \multiply\count3 by 9 %
3021 \advance\count11 by \count3 %

```

```

3022     \loop
3023         % (\count2, \count3) <- (\count2 div 2, \count2 mod 2)
3024         \count3\count2 %
3025         \divide\count3 by 2 %
3026         \multiply\count3 by 2 %
3027         \count3-\count3 %
3028         \advance\count3 by \count2 %
3029         \divide\count2 by 2 %
3030         \ifnum\count3=1 %
3031             \let\@tempg\@tempe
3032             \edef\@tempe{\@tempg\@tempf}%
3033         \fi
3034         \let\@tempg\@tempf
3035         \edef\@tempf{\@tempg\@tempg}%
3036         \ifnum\count2>0 %
3037     \repeat
3038 \fi
3039 \divide\count11 by 3 %
3040 \ifcase\count11 % 0 .. 5
3041 % 0 => d milliard(s) (de milliards)*
3042 \def\@temph{2}%
3043 \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space
3044 \or % 1 => d mille milliard(s) (de milliards)*
3045 \def\@temph{1}%
3046 \count10=\fc@frenchoptions@mil@plural\space
3047 \or % 2 => d million(s) (de milliards)*
3048 \def\@temph{2}%
3049 \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
3050 \or % 3 => d milliard(s) (de milliards)*
3051 \def\@temph{2}%
3052 \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space
3053 \or % 4 => d mille milliards (de milliards)*
3054 \def\@temph{1}%
3055 \count10=\fc@frenchoptions@mil@plural\space
3056 \else % 5 => d million(s) (de milliards)*
3057 \def\@temph{2}%
3058 \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
3059 \fi
3060 \let\@tempg\@tempe
3061 \edef\@tempf{%
3062     \ifcase\count11% 0 .. 5
3063     \or
3064         mil\fc@@do@plural@mark \fc@frenchoptions@mil@plural@mark
3065     \or
3066         million\fc@@do@plural@mark s%
3067     \or
3068         milliard\fc@@do@plural@mark s%
3069     \or
3070         mil\fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark

```

```

3071         \noexpand\@nil\fc@frenchoptions@supermillion@dos
3072         \noexpand\fc@case milliards% 4
3073     \or
3074         million\fc@@do@plural@mark s%
3075         \noexpand\@nil\fc@frenchoptions@supermillion@dos
3076         de\fc@frenchoptions@supermillion@dos\noexpand\fc@case milliards% 5
3077     \fi
3078 }%
3079 \edef\@tempe{%
3080     \ifx\@tempf\@empty\else
3081         \expandafter\fc@case\@tempf\@nil
3082     \fi
3083     \@tempg
3084 }%
3085 \else
3086     \def\@temph{0}%
3087     \let\@tempe\@empty
3088 \fi

```

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

```

3089     \expandafter\toks\expandafter1\expandafter{\@tempe}%
3090     \toks0{#2}%
3091     \edef\@tempa{\the\toks0 \@temph \def\noexpand#3{\the\toks1}}%
3092     \expandafter
3093 } \@tempa
3094 }

```

`\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.

```

3095 \ifcsundef{fc@muladdfrench}{-}{%
3096     \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
3097     'fc@muladdfrench'}}

```

Arguments as follows:

- #2 input, plural indicator for number d
- #3 input, formatted number d
- #5 input, formatted number 10^w , i.e. power of ten which is multiplied by d

Implicit arguments from context:

`\@tempa` input, formatted number a
output, macro to which place the mul-add result
`\count8` input, power type indicator for $10^{w'}$, where w' is a weight of a , this is an index in $[0..2]$ that reflects whether $10^{w'}$ is formatted by “mil(le)” — for index = 1 — or by “ $\langle n \rangle$ illion(s)| $\langle n \rangle$ illiard(s)” — for index = 2
`\count9` input, power type indicator for 10^w , this is an index in $[0..2]$ that reflect whether the weight w of d is formatted by “metan-othing” — for index = 0, “mil(le)” — for index = 1 — or by “ $\langle n \rangle$ illion(s)| $\langle n \rangle$ illiard(s)” — for index = 2

```

3098 \def\fc@muladdfrench#1#2#3{%
3099   {%

```

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

```

3100   \edef\@tempc{#1}%
3101   \edef\@tempd{#2}%
3102   \edef\@tempf{#3}%
3103   \let\@tempc\@tempc
3104   \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”.

```

3105   \ifnum\@tempc=1 %
3106     \ifnum\count9=1 %
3107       \let\@tempd\empty
3108     \fi
3109   \fi

```

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

```

3110   \edef\@tempg{%
3111     \@tempd
3112     \ifx\@tempd\empty\else
3113       \ifx\@tempf\empty\else
3114         \ifcase\count9 %
3115           \or
3116             \fc@frenchoptions@submillion@dos
3117           \or
3118             \fc@frenchoptions@supermillion@dos
3119           \fi
3120         \fi
3121       \fi
3122     \@tempf
3123   }%

```

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

```

3124 \edef\@temph{%
3125   \@tempa
3126   \ifx\@tempa\@empty\else
3127     \ifx\@tempg\@empty\else
3128       \ifcase\count8 %
3129         \or
3130         \fc@frenchoptions@submillion@dos
3131         \or
3132         \fc@frenchoptions@supermillion@dos
3133       \fi
3134     \fi
3135   \fi
3136   \@tempg
3137 }%

```

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

```

3138 \def\@tempb##1{\def\@tempa{\def\@tempa{##1}}}%
3139 \expandafter\@tempb\expandafter{\@temph}%
3140 \expandafter
3141 }\@tempa
3142 }%

```

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

```

3143 \ifcsundef{fc@lthundredstringfrench}{-}{%
3144   \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
3145     'fc@lthundredstringfrench'}}

```

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

#1 input/output macro to which append the result

Implicit input arguments as follows:

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

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

```

3146 \def\fc@lthundredstringfrench#1{%
3147   {%

```

First save arguments into local temporary macro.

```

3148   \let\@tempc#1%

```

Read units d_w to `\count1`.

```

3149   \fc@read@unit{\count1}{\count0}%

```

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

```

3150   \count3\count0 %
3151   \advance\count3 1 %
3152   \fc@read@unit{\count2}{\count3}%

```

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

```

3153 \edef\@tempa{%
3154   \@tempc
3155   \ifnum\count2>1 %
3156     % 20 .. 99
3157     \ifnum\count2>6 %
3158       % 70 .. 99
3159       \ifnum\count2<8 %
3160         % 70 .. 79
3161         \@seventies{\count1}%
3162       \else
3163         % 80..99
3164         \ifnum\count2<9 %
3165           % 80 .. 89
3166           \@eighties{\count1}%
3167         \else
3168           % 90 .. 99
3169           \@nineties{\count1}%
3170         \fi
3171       \fi
3172     \else
3173       % 20..69
3174       \@tenstring{\count2}%
3175       \ifnum\count1>0 %
3176         % x1 .. x0
3177         \ifnum\count1=1 %
3178           % x1
3179           \fc@frenchoptions@submillion@dos\@andname\fc@frenchoptions@submillion@dos
3180         \else
3181           % x2 .. x9
3182           -%
3183         \fi
3184         \@unitstring{\count1}%
3185       \fi
3186     \fi
3187   \else
3188     % 0 .. 19
3189     \ifnum\count2=0 % when tens = 0
3190       % 0 .. 9
3191       \ifnum\count1=0 % when units = 0
3192         % \count3=1 when #1 = 0, i.e. only for the unit of the top level number
3193         \ifnum\count3=1 %
3194           \ifnum\fc@max@weight=0 %
3195             \@unitstring{0}%
3196           \fi
3197         \fi
3198       \else
3199         % 1 .. 9
3200         \@unitstring{\count1}%

```

```

3201         \fi
3202     \else
3203         % 10 .. 19
3204         \@teenstring{\count1}%
3205     \fi
3206 \fi
3207 }%

```

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

```

3208     \def\@tempb##1{\def\@tempa{\def#1{##1}}}%
3209     \expandafter\@tempb\expandafter{\@tempa}%
3210     \expandafter
3211 } \@tempa
3212 }

```

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

```

3213 \ifcsundef{fc@ltthousandstringfrench}{}%
3214 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
3215 '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)

```

3216 \def\fc@ltthousandstringfrench#1{%
3217   {%

```

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

```

3218     \count4\count0 %
3219     \advance\count4 by 2 %
3220     \fc@read@unit{\count2 }{\count4 }%

```

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

```

3221     \advance\count4 by -1 %
3222     \count3\count4 %
3223     \advance\count3 by -1 %
3224     \fc@check@nonzeros{\count3 }{\count4 } \@tempa

```

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

```

3225     \edef\@temps{%
3226         \ifcase\fc@frenchoptions@cent@plural\space
3227         % 0 => always
3228         s%
3229         \or
3230         % 1 => never
3231         \or

```

```

3232     % 2 => multiple
3233     \ifnum\count2>1s\fi
3234     \or
3235     % 3 => multiple g-last
3236     \ifnum\count2>1 \ifnum\@tempa=0 \ifnum\count0=\count6s\fi\fi\fi
3237     \or
3238     % 4 => multiple l-last
3239     \ifnum\count2>1 \ifnum\@tempa=0 \ifnum\count9=0s\else\ifnum\count9=2s\fi\fi\fi\fi
3240     \fi
3241 }%
3242 % compute spacing after cent(s?) into \@tempb
3243 \expandafter\let\expandafter\@tempb
3244     \ifnum\@tempa>0 \fc@frenchoptions@submillion@dos\else\@empty\fi
3245 % now place into \@tempa the hundreds
3246 \edef\@tempa{%
3247     \ifnum\count2=0 %
3248     \else
3249     \ifnum\count2=1 %
3250     \expandafter\fc@case\@hundred\@nil
3251     \else
3252     \@unitstring{\count2}\fc@frenchoptions@submillion@dos
3253     \noexpand\fc@case\@hundred\@temps\noexpand\@nil
3254     \fi
3255     \@tempb
3256     \fi
3257 }%
3258 % now append to \@tempa the ten and unit
3259 \fc@lthundredstringfrench\@tempa

    Propagate expansion of \@tempa into macro #1 after closing brace.
3260     \def\@tempb##1{\def\@tempa{\def#1{##1}}}%
3261     \expandafter\@tempb\expandafter{\@tempa}%
3262     \expandafter
3263 } \@tempa
3264 }

```

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

```

3265 \ifcsundef{@@numberstringfrench}{\}%
3266 \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

```

3267 \def\@@numberstringfrench#1#2{%
3268     {%

```

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

```

3269     \edef\@tempa{#1}%
3270     \expandafter\fc@number@parser\expandafter{\@tempa}%

```

```

3271 \ifnum\fc@min@weight<0 %
3272 \PackageError{fmtcount}{Out of range}%
3273 {This macro does not work with fractional numbers}%
3274 \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’.

```

3275 \edef\@tempa{\ifcase\fc@sign@case\space\or\fc@case plus\@nil\or\fc@case moins\@nil\fi}%
3276 \fc@nbrstr@preamble
3277 \fc@@nbrstrfrench@inner
3278 \fc@nbrstr@postamble

```

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

```

3279 \def\@tempb##1{\def\@tempa{\def#2{##1}}}%
3280 \expandafter\@tempb\expandafter{\@tempa}%
3281 \expandafter
3282 }\@tempa
3283 }

```

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

Arguments are as follows:

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

```

3284 \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.

```

3285 \count0=\fc@max@weight
3286 \divide\count0 by 3 %
3287 \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.

```

3288 \fc@intpart@find@last{\count5 }%
3289 \count6\count5 %
3290 \divide\count6 3 %
3291 \multiply\count6 3 %
3292 \count8=0 %
3293 \loop

```

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

```

3294 \count1\count0 %
3295 \advance\count1 by 2 %
3296 \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.

```
3297 \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

```
3298 \fc@1tthousandstringfrench\@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 .

```
3299 \fc@muladdfrench\@tempt\@tempd\@tempb
```

Then iterate.

```
3300 \count8\count9 %
```

```
3301 \advance\count0 by -3 %
```

```
3302 \ifnum\count6>\count0 \else
```

```
3303 \repeat
```

```
3304 }
```

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

```
3305 \ifcsundef{@@ordinalstringfrench}{-}{%
```

```
3306 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
```

```
3307 '@@ordinalstringfrench'}}
```

Arguments are as follows:

#1 number to convert to string

#2 macro into which to place the result

```
3308 \def\@@ordinalstringfrench#1#2{%
```

```
3309 {%
```

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

```
3310 \edef\@tempa{#1}%
```

```
3311 \expandafter\fc@number@parser\expandafter{\@tempa}%
```

```
3312 \ifnum\fc@min@weight<0 %
```

```
3313 \PackageError{fmtcount}{Out of range}%
```

```
3314 {This macro does not work with fractional numbers}%
```

```
3315 \fi
```

```
3316 \ifnum\fc@sign@case>0 %
```

```
3317 \PackageError{fmtcount}{Out of range}%
```

```
3318 {This macro does with negative or explicitly marked as positive numbers}%
```

```
3319 \fi
```

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

```
3320 \ifnum\fc@max@weight=0 %
```

```
3321 \ifnum\csname fc@digit@0\endcsname=1 %
```

```
3322 \count0=1 %
```

```
3323 \else
```

```
3324 \count0=0 %
```

```

3325     \fi
3326   \else
3327     \count0=0 %
3328   \fi
3329   \ifnum\count0=1 %
3330     \edef\@tempa{\expandafter\fc@case\fc@first\@nil}%
3331   \else

```

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

```

3332     \def\@tempa##1{%
3333       \expandafter\edef\csname fc@frenchoptions@##1@plural\endcsname{%
3334         \ifcase\csname fc@frenchoptions@##1@plural\endcsname\space
3335         0% 0: always => always
3336         \or
3337         1% 1: never => never
3338         \or
3339         6% 2: multiple => multiple ng-last
3340         \or
3341         1% 3: multiple g-last => never
3342         \or
3343         5% 4: multiple l-last => multiple lng-last
3344         \or
3345         5% 5: multiple lng-last => multiple lng-last
3346         \or
3347         6% 6: multiple ng-last => multiple ng-last
3348       \fi
3349     }%
3350   }%
3351   \@tempa{vingt}%
3352   \@tempa{cent}%
3353   \@tempa{mil}%
3354   \@tempa{n-illion}%
3355   \@tempa{n-illiard}%

```

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

```

3356     \let\fc@case@save\fc@case
3357     \def\fc@case{\noexpand\fc@case}%
3358     \def\@nil{\noexpand\@nil}%

```

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

```

3359     \let\@tempa\@empty
3360     \fc@nbrstrfrench@inner

```

Now restore \fc@case

```

3361     \let\fc@case\fc@case@save

```

Now we add the “ième” ending

```

3362     \expandafter\fc@get@last@word\expandafter{\@tempa}\@tempb\@tempc
3363     \expandafter\fc@get@last@letter\expandafter{\@tempc}\@tempd\@tempe
3364     \def\@tempf{e}%

```

```

3365     \ifx\@tempe\@tempf
3366         \edef\@tempa{\@tempb\expandafter\fc@case\@tempd i\‘eme\@nil}%
3367     \else
3368         \def\@tempf{q}%
3369         \ifx\@tempe\@tempf
3370             \edef\@tempa{\@tempb\expandafter\fc@case\@tempd qui\‘eme\@nil}%
3371         \else
3372             \def\@tempf{f}%
3373             \ifx\@tempe\@tempf
3374                 \edef\@tempa{\@tempb\expandafter\fc@case\@tempd vi\‘eme\@nil}%
3375             \else
3376                 \edef\@tempa{\@tempb\expandafter\fc@case\@tempc i\‘eme\@nil}%
3377             \fi
3378         \fi
3379     \fi
3380 \fi

```

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

```

3381     \def\@tempb##1{\def\@tempa{\def#2{##1}}}%
3382     \expandafter\@tempb\expandafter{\@tempa}%
3383     \expandafter
3384 } \@tempa
3385 }

```

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

```

3386 \newcommand*\fc@frenchoptions@setdefaults{%
3387   \csname KV@fcfrench@all plural\endcsname{reformed}%
3388   \def\fc@frenchoptions@submillion@dos{-}%
3389   \let\fc@frenchoptions@supermillion@dos\space
3390   \let\fc@u@in@duo\@empty% Could be ‘u’
3391   % \let\fc@poweroften\fc@pot@longscalefrench
3392   \let\fc@poweroften\fc@pot@recursivefrench
3393   \def\fc@longscale@illiard@upto{0}% infinity
3394   \def\fc@frenchoptions@mil@plural@mark{le}%
3395 }
3396 \fc@frenchoptions@setdefaults

```

9.4.6 fc-frenchb.def

```

3397 \ProvidesFCLanguage{frenchb}[2013/08/17]%
3398 \FCLoadlang{french}%

```

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

```

3399 \global\let\@ordinalMfrenchb=\@ordinalMfrench
3400 \global\let\@ordinalFfrenchb=\@ordinalFfrench
3401 \global\let\@ordinalNfrenchb=\@ordinalNfrench
3402 \global\let\@numberstringMfrenchb=\@numberstringMfrench
3403 \global\let\@numberstringFfrenchb=\@numberstringFfrench

```

```

3404 \global\let\@numberstringNfrenchb=\@numberstringNfrench
3405 \global\let\@NumberstringMfrenchb=\@NumberstringMfrench
3406 \global\let\@NumberstringFfrenchb=\@NumberstringFfrench
3407 \global\let\@NumberstringNfrenchb=\@NumberstringNfrench
3408 \global\let\@ordinalstringMfrenchb=\@ordinalstringMfrench
3409 \global\let\@ordinalstringFfrenchb=\@ordinalstringFfrench
3410 \global\let\@ordinalstringNfrenchb=\@ordinalstringNfrench
3411 \global\let\@OrdinalstringMfrenchb=\@OrdinalstringMfrench
3412 \global\let\@OrdinalstringFfrenchb=\@OrdinalstringFfrench
3413 \global\let\@OrdinalstringNfrenchb=\@OrdinalstringNfrench

```

9.4.7 fc-german.def

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

```
3414 \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:

```

3415 \newcommand{\@ordinalMgerman}[2]{%
3416   \edef#2{\number#1\relax.}%
3417 }%
3418 \global\let\@ordinalMgerman\@ordinalMgerman

```

Feminine:

```

3419 \newcommand{\@ordinalFgerman}[2]{%
3420   \edef#2{\number#1\relax.}%
3421 }%
3422 \global\let\@ordinalFgerman\@ordinalFgerman

```

Neuter:

```

3423 \newcommand{\@ordinalNgerman}[2]{%
3424   \edef#2{\number#1\relax.}%
3425 }%
3426 \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):

```

3427 \newcommand*\@unitstringgerman[1]{%
3428   \ifcase#1%
3429     null%
3430     \or ein%
3431     \or zwei%
3432     \or drei%
3433     \or vier%
3434     \or f\"unf%
3435     \or sechs%
3436     \or sieben%
3437     \or acht%
3438     \or neun%

```

```

3439 \fi
3440 }%
3441 \global\let\@@unitstringgerman\@@unitstringgerman

```

Tens (argument must go from 1 to 10):

```

3442 \newcommand*\@@tenstringgerman[1]{%
3443   \ifcase#1%
3444     \or zehn%
3445     \or zwanzig%
3446     \or drei{\ss}ig%
3447     \or vierzig%
3448     \or f\unfzig%
3449     \or sechzig%
3450     \or siebzig%
3451     \or achtzig%
3452     \or neunzig%
3453     \or einhundert%
3454   \fi
3455 }%
3456 \global\let\@@tenstringgerman\@@tenstringgerman

```

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

```

3457 \providecommand*\einhundert{einhundert}%
3458 \providecommand*\eintausend{eintausend}%
3459 \global\let\einhundert\einhundert
3460 \global\let\eintausend\eintausend

```

Teens:

```

3461 \newcommand*\@@teenstringgerman[1]{%
3462   \ifcase#1%
3463     zehn%
3464     \or elf%
3465     \or zw\olf%
3466     \or dreizehn%
3467     \or vierzehn%
3468     \or f\unfzehn%
3469     \or sechzehn%
3470     \or siebzehn%
3471     \or achtzehn%
3472     \or neunzehn%
3473   \fi
3474 }%
3475 \global\let\@@teenstringgerman\@@teenstringgerman

```

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

```

3476 \DeclareRobustCommand{\@numberstringMgerman}[2]{%
3477   \let\@unitstring=\@@unitstringgerman
3478   \let\@teenstring=\@@teenstringgerman
3479   \let\@tenstring=\@@tenstringgerman
3480   \@numberstringgerman{#1}{#2}%

```

```

3481 }%
3482 \global\let\@numberstringMgerman\@numberstringMgerman

  Feminine and neuter forms:
3483 \global\let\@numberstringFgerman=\@numberstringMgerman
3484 \global\let\@numberstringNgerman=\@numberstringMgerman

  As above, but initial letters in upper case:
3485 \DeclareRobustCommand{\@NumberstringMgerman}[2]{%
3486   \@numberstringMgerman{#1}{\@num@str}%
3487   \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
3488 }%
3489 \global\let\@NumberstringMgerman\@NumberstringMgerman

  Feminine and neuter form:
3490 \global\let\@NumberstringFgerman=\@NumberstringMgerman
3491 \global\let\@NumberstringNgerman=\@NumberstringMgerman

  As above, but for ordinals.
3492 \DeclareRobustCommand{\@OrdinalstringMgerman}[2]{%
3493   \let\@unitthstring=\@unitthstringMgerman
3494   \let\@teenthstring=\@teenthstringMgerman
3495   \let\@tenthstring=\@tenthstringMgerman
3496   \let\@unitstring=\@unitstringgerman
3497   \let\@teenstring=\@teenstringgerman
3498   \let\@tenstring=\@tenstringgerman
3499   \def\@thousandth{tausendster}%
3500   \def\@hundredth{hundertster}%
3501   \@Ordinalstringgerman{#1}{#2}%
3502 }%
3503 \global\let\@OrdinalstringMgerman\@OrdinalstringMgerman

  Feminine form:
3504 \DeclareRobustCommand{\@OrdinalstringFgerman}[2]{%
3505   \let\@unitthstring=\@unitthstringFgerman
3506   \let\@teenthstring=\@teenthstringFgerman
3507   \let\@tenthstring=\@tenthstringFgerman
3508   \let\@unitstring=\@unitstringgerman
3509   \let\@teenstring=\@teenstringgerman
3510   \let\@tenstring=\@tenstringgerman
3511   \def\@thousandth{tausendste}%
3512   \def\@hundredth{hundertste}%
3513   \@Ordinalstringgerman{#1}{#2}%
3514 }%
3515 \global\let\@OrdinalstringFgerman\@OrdinalstringFgerman

  Neuter form:
3516 \DeclareRobustCommand{\@OrdinalstringNgerman}[2]{%
3517   \let\@unitthstring=\@unitthstringNgerman
3518   \let\@teenthstring=\@teenthstringNgerman
3519   \let\@tenthstring=\@tenthstringNgerman
3520   \let\@unitstring=\@unitstringgerman

```

```

3521 \let\@teenstring=\@teenstringgerman
3522 \let\@tenstring=\@tenstringgerman
3523 \def\@thousandth{tausendstes}%
3524 \def\@hundredth{hunderstes}%
3525 \@ordinalstringgerman{#1}{#2}%
3526 }%
3527 \global\let\@ordinalstringNgerman\@ordinalstringNgerman

```

As above, but with initial letters in upper case.

```

3528 \DeclareRobustCommand{\@OrdinalstringMgerman}[2]{%
3529 \@ordinalstringMgerman{#1}{\@num@str}%
3530 \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
3531 }%
3532 \global\let\@OrdinalstringMgerman\@OrdinalstringMgerman

```

Feminine form:

```

3533 \DeclareRobustCommand{\@OrdinalstringFgerman}[2]{%
3534 \@ordinalstringFgerman{#1}{\@num@str}%
3535 \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
3536 }%
3537 \global\let\@OrdinalstringFgerman\@OrdinalstringFgerman

```

Neuter form:

```

3538 \DeclareRobustCommand{\@OrdinalstringNgerman}[2]{%
3539 \@ordinalstringNgerman{#1}{\@num@str}%
3540 \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
3541 }%
3542 \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:

```

3543 \newcommand*\@unitthstringMgerman[1]{%
3544 \ifcase#1%
3545 nullter%
3546 \or erster%
3547 \or zweiter%
3548 \or dritter%
3549 \or vierter%
3550 \or f"unfter%
3551 \or sechster%
3552 \or siebter%
3553 \or achter%
3554 \or neunter%
3555 \fi
3556 }%
3557 \global\let\@unitthstringMgerman\@unitthstringMgerman

```

Tens:

```

3558 \newcommand*\@tenthstringMgerman[1]{%
3559 \ifcase#1%
3560 \or zehnter%

```

```

3561 \or zwanzigster%
3562 \or drei{\ss}igster%
3563 \or vierzigster%
3564 \or f\"unfzigster%
3565 \or sechzigster%
3566 \or siebzigster%
3567 \or achtzigster%
3568 \or neunzigster%
3569 \fi
3570 }%
3571 \global\let\@@tenthstringMgerman\@@tenthstringMgerman

```

Teens:

```

3572 \newcommand*\@@teenthstringMgerman[1]{%
3573 \ifcase#1%
3574 zehnter%
3575 \or elfter%
3576 \or zw\"olfte%
3577 \or dreizehnter%
3578 \or vierzehnter%
3579 \or f\"unfzehnter%
3580 \or sechzehnter%
3581 \or siebzehnter%
3582 \or achtzehnter%
3583 \or neunzehnter%
3584 \fi
3585 }%
3586 \global\let\@@teenthstringMgerman\@@teenthstringMgerman

```

Units (feminine):

```

3587 \newcommand*\@@unitthstringFgerman[1]{%
3588 \ifcase#1%
3589 nullte%
3590 \or erste%
3591 \or zweite%
3592 \or dritte%
3593 \or vierte%
3594 \or f\"unfte%
3595 \or sechste%
3596 \or siebte%
3597 \or achte%
3598 \or neunte%
3599 \fi
3600 }%
3601 \global\let\@@unitthstringFgerman\@@unitthstringFgerman

```

Tens (feminine):

```

3602 \newcommand*\@@tenthstringFgerman[1]{%
3603 \ifcase#1%
3604 \or zehnte%
3605 \or zwanzigste%

```

```

3606 \or drei{\ss}igste%
3607 \or vierzigste%
3608 \or f\"unfzigste%
3609 \or sechzigste%
3610 \or siebzigste%
3611 \or achtzigste%
3612 \or neunzigste%
3613 \fi
3614 }%
3615 \global\let\@tenthstringFgerman\@tenthstringFgerman

    Teens (feminine)
3616 \newcommand*\@teenthstringFgerman[1]{%
3617 \ifcase#1%
3618 zehnte%
3619 \or elfte%
3620 \or zw\"olfte%
3621 \or dreizehnte%
3622 \or vierzehnte%
3623 \or f\"unfzehnte%
3624 \or sechzehnte%
3625 \or siebzehnte%
3626 \or achtzehnte%
3627 \or neunzehnte%
3628 \fi
3629 }%
3630 \global\let\@teenthstringFgerman\@teenthstringFgerman

    Units (neuter):
3631 \newcommand*\@unitthstringNgerman[1]{%
3632 \ifcase#1%
3633 nulltes%
3634 \or erstes%
3635 \or zweites%
3636 \or drittes%
3637 \or viertes%
3638 \or f\"unftes%
3639 \or sechstes%
3640 \or siebtes%
3641 \or achttes%
3642 \or neuntes%
3643 \fi
3644 }%
3645 \global\let\@unitthstringNgerman\@unitthstringNgerman

    Tens (neuter):
3646 \newcommand*\@tenthstringNgerman[1]{%
3647 \ifcase#1%
3648 \or zehntes%
3649 \or zwanzigstes%
3650 \or drei{\ss}igstes%

```

```

3651 \or vierzigstes%
3652 \or f\"unfzigstes%
3653 \or sechzigstes%
3654 \or siebzigstes%
3655 \or achtzigstes%
3656 \or neunzigstes%
3657 \fi
3658 }%
3659 \global\let\@@tenthstringNgerman\@@tenthstringNgerman

```

Teens (neuter)

```

3660 \newcommand*\@@teenthstringNgerman[1]{%
3661 \ifcase#1%
3662 zehntes%
3663 \or elftes%
3664 \or zw\"olftes%
3665 \or dreizehntes%
3666 \or vierzehntes%
3667 \or f\"unfzehntes%
3668 \or sechzehntes%
3669 \or siebzehntes%
3670 \or achtzehntes%
3671 \or neunzehntes%
3672 \fi
3673 }%
3674 \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.

```

3675 \newcommand*\@@numberunderhundredgerman[2]{%
3676 \ifnum#1<10\relax
3677 \ifnum#1>0\relax
3678 \eappto#2{\@unitstring{#1}}%
3679 \fi
3680 \else
3681 \@tmpstrctr=#1\relax
3682 \@FCmodulo{\@tmpstrctr}{10}%
3683 \ifnum#1<20\relax
3684 \eappto#2{\@teenstring{\@tmpstrctr}}%
3685 \else
3686 \ifnum\@tmpstrctr=0\relax
3687 \else
3688 \eappto#2{\@unitstring{\@tmpstrctr}und}%
3689 \fi
3690 \@tmpstrctr=#1\relax
3691 \divide\@tmpstrctr by 10\relax
3692 \eappto#2{\@tenstring{\@tmpstrctr}}%
3693 \fi
3694 \fi
3695 }%

```

```

3696 \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.
3697 \newcommand*\@@numberstringgerman[2]{%
3698 \ifnum#1>99999\relax
3699   \PackageError{fmtcount}{Out of range}%
3700   {This macro only works for values less than 100000}%
3701 \else
3702   \ifnum#1<0\relax
3703     \PackageError{fmtcount}{Negative numbers not permitted}%
3704     {This macro does not work for negative numbers, however
3705     you can try typing "minus" first, and then pass the modulus of
3706     this number}%
3707   \fi
3708 \fi
3709 \def#2{}%
3710 \@strctr=#1\relax \divide\@strctr by 1000\relax
3711 \ifnum\@strctr>1\relax
    #1 is  $\geq 2000$ , \@strctr now contains the number of thousands
3712 \@@numberunderhundredgerman{\@strctr}{#2}%
3713   \appto#2{tausend}%
3714 \else
    #1 lies in range [1000,1999]
3715   \ifnum\@strctr=1\relax
3716     \eappto#2{eintausend}%
3717   \fi
3718 \fi
3719 \@strctr=#1\relax
3720 \@FCmodulo{\@strctr}{1000}%
3721 \divide\@strctr by 100\relax
3722 \ifnum\@strctr>1\relax
    now dealing with number in range [200,999]
3723   \eappto#2{\@unitstring{\@strctr}hundert}%
3724 \else
3725   \ifnum\@strctr=1\relax
    dealing with number in range [100,199]
3726     \ifnum#1>1000\relax
    if original number > 1000, use einhundert
3727       \appto#2{einhundert}%
3728     \else
    otherwise use \einhundert
3729       \eappto#2{\einhundert}%
3730     \fi
3731   \fi
3732 \fi

```

```

3733 \@strctr=#1\relax
3734 \@FCmodulo{\@strctr}{100}%
3735 \ifnum#1=0\relax
3736   \def#2{null}%
3737 \else
3738   \ifnum\@strctr=1\relax
3739     \appto#2{eins}%
3740   \else
3741     \@numberunderhundredgerman{\@strctr}{#2}%
3742   \fi
3743 \fi
3744 }%
3745 \global\let\@numberstringgerman\@numberstringgerman

```

As above, but for ordinals

```

3746 \newcommand*\@numberunderhundredthgerman[2]{%
3747 \ifnum#1<10\relax
3748   \eappto#2{\@unitthstring{#1}}%
3749 \else
3750   \@tmpstrctr=#1\relax
3751   \@FCmodulo{\@tmpstrctr}{10}%
3752   \ifnum#1<20\relax
3753     \eappto#2{\@teenthstring{\@tmpstrctr}}%
3754   \else
3755     \ifnum\@tmpstrctr=0\relax
3756     \else
3757       \eappto#2{\@unitstring{\@tmpstrctr}und}%
3758     \fi
3759     \@tmpstrctr=#1\relax
3760     \divide\@tmpstrctr by 10\relax
3761     \eappto#2{\@tenthstring{\@tmpstrctr}}%
3762   \fi
3763 \fi
3764 }%
3765 \global\let\@numberunderhundredthgerman\@numberunderhundredthgerman

3766 \newcommand*\@ordinalstringgerman[2]{%
3767 \ifnum#1>99999\relax
3768   \PackageError{fmtcount}{Out of range}%
3769   {This macro only works for values less than 100000}%
3770 \else
3771   \ifnum#1<0\relax
3772     \PackageError{fmtcount}{Negative numbers not permitted}%
3773     {This macro does not work for negative numbers, however
3774     you can try typing "minus" first, and then pass the modulus of
3775     this number}%
3776   \fi
3777 \fi
3778 \def#2{}%
3779 \@strctr=#1\relax \divide\@strctr by 1000\relax

```

```

3780 \ifnum\@strctr>1\relax

  #1 is  $\geq 2000$ , \@strctr now contains the number of thousands
3781 \@@numberunderhundredgerman{\@strctr}{#2}%

  is that it, or is there more?
3782 \@tmpstrctr=#1\relax \@FCmodulo{\@tmpstrctr}{1000}%
3783 \ifnum\@tmpstrctr=0\relax
3784 \eappto#2{\@thousandth}%
3785 \else
3786 \appto#2{tausend}%
3787 \fi
3788 \else

  #1 lies in range [1000,1999]
3789 \ifnum\@strctr=1\relax
3790 \ifnum#1=1000\relax
3791 \eappto#2{\@thousandth}%
3792 \else
3793 \eappto#2{\eintausend}%
3794 \fi
3795 \fi
3796 \fi
3797 \@strctr=#1\relax
3798 \@FCmodulo{\@strctr}{1000}%
3799 \divide\@strctr by 100\relax
3800 \ifnum\@strctr>1\relax

  now dealing with number in range [200,999] is that it, or is there more?
3801 \@tmpstrctr=#1\relax \@FCmodulo{\@tmpstrctr}{100}%
3802 \ifnum\@tmpstrctr=0\relax
3803 \ifnum\@strctr=1\relax
3804 \eappto#2{\@hundredth}%
3805 \else
3806 \eappto#2{\@unitstring{\@strctr}\@hundredth}%
3807 \fi
3808 \else
3809 \eappto#2{\@unitstring{\@strctr}hundert}%
3810 \fi
3811 \else
3812 \ifnum\@strctr=1\relax

  dealing with number in range [100,199] is that it, or is there more?
3813 \@tmpstrctr=#1\relax \@FCmodulo{\@tmpstrctr}{100}%
3814 \ifnum\@tmpstrctr=0\relax
3815 \eappto#2{\@hundredth}%
3816 \else
3817 \ifnum#1>1000\relax
3818 \appto#2{einhundert}%
3819 \else
3820 \eappto#2{\einhundert}%

```

```

3821     \fi
3822     \fi
3823   \fi
3824 \fi
3825 \@strctr=#1\relax
3826 \@FCmodulo{\@strctr}{100}%
3827 \ifthenelse{\@strctr=0 \and #1>0}{\}{%
3828 \@numberunderhundredthgerman{\@strctr}{#2}%
3829 }%
3830 }%
3831 \global\let\@@ordinalstringgerman\@@ordinalstringgerman

  Load fc-germanb.def if not already loaded
3832 \FCloadlang{germanb}%

```

9.4.8 fc-germanb.def

```

3833 \ProvidesFCLanguage{germanb}[2013/08/17]%

```

Load fc-german.def if not already loaded

```

3834 \FCloadlang{german}%

```

Set germanb to be equivalent to german.

```

3835 \global\let\@ordinalMgermanb=\@ordinalMgerman
3836 \global\let\@ordinalFgermanb=\@ordinalFgerman
3837 \global\let\@ordinalNgermanb=\@ordinalNgerman
3838 \global\let\@numberstringMgermanb=\@numberstringMgerman
3839 \global\let\@numberstringFgermanb=\@numberstringFgerman
3840 \global\let\@numberstringNgermanb=\@numberstringNgerman
3841 \global\let\@NumberstringMgermanb=\@NumberstringMgerman
3842 \global\let\@NumberstringFgermanb=\@NumberstringFgerman
3843 \global\let\@NumberstringNgermanb=\@NumberstringNgerman
3844 \global\let\@ordinalstringMgermanb=\@ordinalstringMgerman
3845 \global\let\@ordinalstringFgermanb=\@ordinalstringFgerman
3846 \global\let\@ordinalstringNgermanb=\@ordinalstringNgerman
3847 \global\let\@OrdinalstringMgermanb=\@OrdinalstringMgerman
3848 \global\let\@OrdinalstringFgermanb=\@OrdinalstringFgerman
3849 \global\let\@OrdinalstringNgermanb=\@OrdinalstringNgerman

```

9.4.9 fc-italian

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

```

3850 \ProvidesFCLanguage{italian}[2013/08/17]
3851
3852 \RequirePackage{itnumpar}
3853
3854 \newcommand{\@numberstringMitalian}[2]{%
3855   \edef#2{\noexpand\printnumeroinparole{#1}}%
3856 }
3857 \global\let\@numberstringMitalian\@numberstringMitalian

```

```

3858
3859 \newcommand{\@numberstringFitalian}[2]{%
3860   \edef#2{\noexpand\printnumeroinparole{#1}}}
3861
3862 \global\let\@numberstringFitalian\@numberstringFitalian
3863
3864 \newcommand{\@NumberstringMitalian}[2]{%
3865   \edef#2{\noexpand\printNumeroinparole{#1}}%
3866 }
3867 \global\let\@NumberstringMitalian\@NumberstringMitalian
3868
3869 \newcommand{\@NumberstringFitalian}[2]{%
3870   \edef#2{\noexpand\printNumeroinparole{#1}}%
3871 }
3872 \global\let\@NumberstringFitalian\@NumberstringFitalian
3873
3874 \newcommand{\@ordinalstringMitalian}[2]{%
3875   \edef#2{\noexpand\printordinalem{#1}}%
3876 }
3877 \global\let\@ordinalstringMitalian\@ordinalstringMitalian
3878
3879 \newcommand{\@ordinalstringFitalian}[2]{%
3880   \edef#2{\noexpand\printordinalef{#1}}%
3881 }
3882 \global\let\@ordinalstringFitalian\@ordinalstringFitalian
3883
3884 \newcommand{\@OrdinalstringMitalian}[2]{%
3885   \edef#2{\noexpand\printOrdinalem{#1}}%
3886 }
3887 \global\let\@OrdinalstringMitalian\@OrdinalstringMitalian
3888
3889 \newcommand{\@OrdinalstringFitalian}[2]{%
3890   \edef#2{\noexpand\printOrdinalef{#1}}%
3891 }
3892 \global\let\@OrdinalstringFitalian\@OrdinalstringFitalian
3893
3894 \newcommand{\@ordinalMitalian}[2]{%
3895   \edef#2{#1\relax\noexpand\fmtord{o}}}
3896
3897 \global\let\@ordinalMitalian\@ordinalMitalian
3898
3899 \newcommand{\@ordinalFitalian}[2]{%
3900   \edef#2{#1\relax\noexpand\fmtord{a}}}
3901 \global\let\@ordinalFitalian\@ordinalFitalian

```

9.4.10 fc-ngerman.def

```

3902 \ProvidesFCLanguage{ngerman}[2012/06/18]%
3903 \FCloadlang{german}%

```

```
3904 \FCloadlang{ngermanb}%
```

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

```
3905 \global\let\@ordinalMngerman=\@ordinalMgerman
3906 \global\let\@ordinalFngerman=\@ordinalFgerman
3907 \global\let\@ordinalNngerman=\@ordinalNgerman
3908 \global\let\@numberstringMngerman=\@numberstringMgerman
3909 \global\let\@numberstringFngerman=\@numberstringFgerman
3910 \global\let\@numberstringNngerman=\@numberstringNgerman
3911 \global\let\@NumberstringMngerman=\@NumberstringMgerman
3912 \global\let\@NumberstringFngerman=\@NumberstringFgerman
3913 \global\let\@NumberstringNngerman=\@NumberstringNgerman
3914 \global\let\@ordinalstringMngerman=\@ordinalstringMgerman
3915 \global\let\@ordinalstringFngerman=\@ordinalstringFgerman
3916 \global\let\@ordinalstringNngerman=\@ordinalstringNgerman
3917 \global\let\@OrdinalstringMngerman=\@OrdinalstringMgerman
3918 \global\let\@OrdinalstringFngerman=\@OrdinalstringFgerman
3919 \global\let\@OrdinalstringNngerman=\@OrdinalstringNgerman
```

9.4.11 fc-ngermanb.def

```
3920 \ProvidesFCLanguage{ngermanb}[2013/08/17]%
3921 \FCloadlang{german}%
```

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

```
3922 \global\let\@ordinalMngermanb=\@ordinalMgerman
3923 \global\let\@ordinalFngermanb=\@ordinalFgerman
3924 \global\let\@ordinalNngermanb=\@ordinalNgerman
3925 \global\let\@numberstringMngermanb=\@numberstringMgerman
3926 \global\let\@numberstringFngermanb=\@numberstringFgerman
3927 \global\let\@numberstringNngermanb=\@numberstringNgerman
3928 \global\let\@NumberstringMngermanb=\@NumberstringMgerman
3929 \global\let\@NumberstringFngermanb=\@NumberstringFgerman
3930 \global\let\@NumberstringNngermanb=\@NumberstringNgerman
3931 \global\let\@ordinalstringMngermanb=\@ordinalstringMgerman
3932 \global\let\@ordinalstringFngermanb=\@ordinalstringFgerman
3933 \global\let\@ordinalstringNngermanb=\@ordinalstringNgerman
3934 \global\let\@OrdinalstringMngermanb=\@OrdinalstringMgerman
3935 \global\let\@OrdinalstringFngermanb=\@OrdinalstringFgerman
3936 \global\let\@OrdinalstringNngermanb=\@OrdinalstringNgerman
```

Load fc-ngerman.def if not already loaded

```
3937 \FCloadlang{ngerman}%
```

9.4.12 fc-portuges.def

Portuguse definitions

```
3938 \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:

```

3939 \newcommand*\@ordinalMportuges[2]{%
3940   \ifnum#1=0\relax
3941     \edef#2{\number#1}%
3942   \else
3943     \edef#2{\number#1\relax\noexpand\fmtord{o}}%
3944   \fi
3945 }%
3946 \global\let\@ordinalMportuges\@ordinalMportuges

```

Feminine:

```

3947 \newcommand*\@ordinalFportuges[2]{%
3948   \ifnum#1=0\relax
3949     \edef#2{\number#1}%
3950   \else
3951     \edef#2{\number#1\relax\noexpand\fmtord{a}}%
3952   \fi
3953 }%
3954 \global\let\@ordinalFportuges\@ordinalFportuges

```

Make neuter same as masculine:

```

3955 \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):

```

3956 \newcommand*\@@unitstringportuges[1]{%
3957   \ifcase#1\relax
3958     zero%
3959   \or um%
3960   \or dois%
3961   \or tr\^es%
3962   \or quatro%
3963   \or cinco%
3964   \or seis%
3965   \or sete%
3966   \or oito%
3967   \or nove%
3968   \fi
3969 }%
3970 \global\let\@@unitstringportuges\@@unitstringportuges
3971 % \end{macrocode}
3972 % As above, but for feminine:
3973 % \begin{macrocode}
3974 \newcommand*\@@unitstringFportuges[1]{%
3975   \ifcase#1\relax
3976     zero%
3977     \or uma%

```

```

3978 \or duas%
3979 \or tr\^es%
3980 \or quatro%
3981 \or cinco%
3982 \or seis%
3983 \or sete%
3984 \or oito%
3985 \or nove%
3986 \fi
3987 }%
3988 \global\let\@@unitstringFportuges\@@unitstringFportuges

```

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

```

3989 \newcommand*\@@tenstringportuges[1]{%
3990 \ifcase#1\relax
3991 \or dez%
3992 \or vinte%
3993 \or trinta%
3994 \or quarenta%
3995 \or cinq\ "uenta%
3996 \or sessenta%
3997 \or setenta%
3998 \or oitenta%
3999 \or noventa%
4000 \or cem%
4001 \fi
4002 }%
4003 \global\let\@@tenstringportuges\@@tenstringportuges

```

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

```

4004 \newcommand*\@@teenstringportuges[1]{%
4005 \ifcase#1\relax
4006 dez%
4007 \or onze%
4008 \or doze%
4009 \or treze%
4010 \or quatorze%
4011 \or quinze%
4012 \or dezesseis%
4013 \or dezessete%
4014 \or dezoito%
4015 \or dezenove%
4016 \fi
4017 }%
4018 \global\let\@@teenstringportuges\@@teenstringportuges

```

Hundreds:

```

4019 \newcommand*\@@hundredstringportuges[1]{%
4020 \ifcase#1\relax
4021 \or cento%
4022 \or duzentos%

```

```

4023 \or trezentos%
4024 \or quatrocentos%
4025 \or quinhentos%
4026 \or seiscentos%
4027 \or setecentos%
4028 \or oitocentos%
4029 \or novecentos%
4030 \fi
4031 }%
4032 \global\let\@@hundredstringportuges\@@hundredstringportuges

Hundreds (feminine):
4033 \newcommand*\@@hundredstringFportuges[1]{%
4034 \ifcase#1\relax
4035 \or cento%
4036 \or duzentas%
4037 \or trezentas%
4038 \or quatrocentas%
4039 \or quinhentas%
4040 \or seiscentas%
4041 \or setecentas%
4042 \or oitocentas%
4043 \or novecentas%
4044 \fi
4045 }%
4046 \global\let\@@hundredstringFportuges\@@hundredstringFportuges

Units (initial letter in upper case):
4047 \newcommand*\@@Unitstringportuges[1]{%
4048 \ifcase#1\relax
4049 Zero%
4050 \or Um%
4051 \or Dois%
4052 \or Tr\^es%
4053 \or Quatro%
4054 \or Cinco%
4055 \or Seis%
4056 \or Sete%
4057 \or Oito%
4058 \or Nove%
4059 \fi
4060 }%
4061 \global\let\@@Unitstringportuges\@@Unitstringportuges

As above, but feminine:
4062 \newcommand*\@@UnitstringFportuges[1]{%
4063 \ifcase#1\relax
4064 Zera%
4065 \or Uma%
4066 \or Duas%
4067 \or Tr\^es%

```

```

4068 \or Quatro%
4069 \or Cinco%
4070 \or Seis%
4071 \or Sete%
4072 \or Oito%
4073 \or Nove%
4074 \fi
4075 }%
4076 \global\let\@@UnitstringFportuges\@@UnitstringFportuges

Tens (with initial letter in upper case):
4077 \newcommand*\@@Tenstringportuges[1]{%
4078 \ifcase#1\relax
4079 \or Dez%
4080 \or Vinte%
4081 \or Trinta%
4082 \or Quarenta%
4083 \or Cinq\ "uenta%
4084 \or Sessenta%
4085 \or Setenta%
4086 \or Oitenta%
4087 \or Noventa%
4088 \or Cem%
4089 \fi
4090 }%
4091 \global\let\@@Tenstringportuges\@@Tenstringportuges

Teens (with initial letter in upper case):
4092 \newcommand*\@@Teenstringportuges[1]{%
4093 \ifcase#1\relax
4094 Dez%
4095 \or Onze%
4096 \or Doze%
4097 \or Treze%
4098 \or Quatorze%
4099 \or Quinze%
4100 \or Dezesesseis%
4101 \or Dezessete%
4102 \or Dezoito%
4103 \or Dezenove%
4104 \fi
4105 }%
4106 \global\let\@@Teenstringportuges\@@Teenstringportuges

Hundreds (with initial letter in upper case):
4107 \newcommand*\@@Hundredstringportuges[1]{%
4108 \ifcase#1\relax
4109 \or Cento%
4110 \or Duzentos%
4111 \or Trezentos%
4112 \or Quatrocentos%

```

```

4113 \or Quinhentos%
4114 \or Seiscentos%
4115 \or Setecentos%
4116 \or Oitocentos%
4117 \or Novecentos%
4118 \fi
4119 }%
4120 \global\let\@@Hundredstringportuges\@@Hundredstringportuges

```

As above, but feminine:

```

4121 \newcommand*\@@HundredstringFportuges[1]{%
4122 \ifcase#1\relax
4123 \or Cento%
4124 \or Duzentas%
4125 \or Trezentas%
4126 \or Quatrocentas%
4127 \or Quinhentas%
4128 \or Seiscentas%
4129 \or Setecentas%
4130 \or Oitocentas%
4131 \or Novecentas%
4132 \fi
4133 }%
4134 \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.)

```

4135 \DeclareRobustCommand{\@numberstringMportuges}[2]{%
4136 \let\@unitstring=\@unitstringportuges
4137 \let\@teenstring=\@teenstringportuges
4138 \let\@tenstring=\@tenstringportuges
4139 \let\@hundredstring=\@hundredstringportuges
4140 \def\@hundred{cem}\def\@thousand{mil}%
4141 \def\@andname{e}%
4142 \@numberstringportuges{#1}{#2}%
4143 }%
4144 \global\let\@numberstringMportuges\@numberstringMportuges

```

As above, but feminine form:

```

4145 \DeclareRobustCommand{\@numberstringFportuges}[2]{%
4146 \let\@unitstring=\@unitstringFportuges
4147 \let\@teenstring=\@teenstringportuges
4148 \let\@tenstring=\@tenstringportuges
4149 \let\@hundredstring=\@hundredstringFportuges
4150 \def\@hundred{cem}\def\@thousand{mil}%
4151 \def\@andname{e}%
4152 \@numberstringportuges{#1}{#2}%
4153 }%

```

```

4154 \global\let\@numberstringFportuges\@numberstringFportuges
    Make neuter same as masculine:
4155 \global\let\@numberstringNportuges\@numberstringMportuges
    As above, but initial letters in upper case:
4156 \DeclareRobustCommand{\@NumberstringMportuges}[2]{%
4157   \let\@unitstring=\@Unitstringportuges
4158   \let\@teenstring=\@Teenstringportuges
4159   \let\@tenstring=\@Tenstringportuges
4160   \let\@hundredstring=\@Hundredstringportuges
4161   \def\@hundred{Cem}\def\@thousand{Mil}%
4162   \def\@andname{e}%
4163   \@numberstringportuges{#1}{#2}%
4164 }%
4165 \global\let\@NumberstringMportuges\@NumberstringMportuges
    As above, but feminine form:
4166 \DeclareRobustCommand{\@NumberstringFportuges}[2]{%
4167   \let\@unitstring=\@UnitstringFportuges
4168   \let\@teenstring=\@Teenstringportuges
4169   \let\@tenstring=\@Tenstringportuges
4170   \let\@hundredstring=\@HundredstringFportuges
4171   \def\@hundred{Cem}\def\@thousand{Mil}%
4172   \def\@andname{e}%
4173   \@numberstringportuges{#1}{#2}%
4174 }%
4175 \global\let\@NumberstringFportuges\@NumberstringFportuges
    Make neuter same as masculine:
4176 \global\let\@NumberstringNportuges\@NumberstringMportuges
    As above, but for ordinals.
4177 \DeclareRobustCommand{\@OrdinalstringMportuges}[2]{%
4178   \let\@unitthstring=\@Unitthstringportuges
4179   \let\@unitstring=\@Unitstringportuges
4180   \let\@teenthstring=\@Teenthstringportuges
4181   \let\@tenthstring=\@Tenthstringportuges
4182   \let\@hundredthstring=\@Hundredthstringportuges
4183   \def\@thousandth{mil\'esimo}%
4184   \@ordinalstringportuges{#1}{#2}%
4185 }%
4186 \global\let\@OrdinalstringMportuges\@OrdinalstringMportuges
    Feminine form:
4187 \DeclareRobustCommand{\@OrdinalstringFportuges}[2]{%
4188   \let\@unitthstring=\@UnitthstringFportuges
4189   \let\@unitstring=\@UnitstringFportuges
4190   \let\@teenthstring=\@Teenthstringportuges
4191   \let\@tenthstring=\@TenthstringFportuges
4192   \let\@hundredthstring=\@HundredthstringFportuges
4193   \def\@thousandth{mil\'esima}%

```

```

4194 \@@ordinalstringportuges{#1}{#2}%
4195 }%
4196 \global\let\@ordinalstringFportuges\@ordinalstringFportuges

  Make neuter same as masculine:
4197 \global\let\@ordinalstringNportuges\@ordinalstringMportuges

  As above, but initial letters in upper case (masculine):
4198 \DeclareRobustCommand{\@OrdinalstringMportuges}[2]{%
4199 \let\@unitthstring=\@Unitthstringportuges
4200 \let\@unitstring=\@Unitstringportuges
4201 \let\@teenthstring=\@Teenthstringportuges
4202 \let\@tenthstring=\@Tenthstringportuges
4203 \let\@hundredthstring=\@Hundredthstringportuges
4204 \def\@thousandth{Mil\ 'esimo}%
4205 \@@ordinalstringportuges{#1}{#2}%
4206 }%
4207 \global\let\@OrdinalstringMportuges\@OrdinalstringMportuges

  Feminine form:
4208 \DeclareRobustCommand{\@OrdinalstringFportuges}[2]{%
4209 \let\@unitthstring=\@UnitthstringFportuges
4210 \let\@unitstring=\@UnitstringFportuges
4211 \let\@teenthstring=\@Teenthstringportuges
4212 \let\@tenthstring=\@TenthstringFportuges
4213 \let\@hundredthstring=\@HundredthstringFportuges
4214 \def\@thousandth{Mil\ 'esima}%
4215 \@@ordinalstringportuges{#1}{#2}%
4216 }%
4217 \global\let\@OrdinalstringFportuges\@OrdinalstringFportuges

  Make neuter same as masculine:
4218 \global\let\@OrdinalstringNportuges\@OrdinalstringMportuges

  In order to do the ordinals, split into units, teens, tens and hundreds. Units:
4219 \newcommand*\@unitthstringportuges[1]{%
4220 \ifcase#1\relax
4221 zero%
4222 \or primeiro%
4223 \or segundo%
4224 \or terceiro%
4225 \or quarto%
4226 \or quinto%
4227 \or sexto%
4228 \or s\ 'etimo%
4229 \or oitavo%
4230 \or nono%
4231 \fi
4232 }%
4233 \global\let\@unitthstringportuges\@unitthstringportuges

  Tens:

```

```

4234 \newcommand*\@@tenthstringportuges[1]{%
4235   \ifcase#1\relax
4236     \or d\'ecimo%
4237     \or vig\'esimo%
4238     \or trig\'esimo%
4239     \or quadrag\'esimo%
4240     \or q\'uinquag\'esimo%
4241     \or sexag\'esimo%
4242     \or setuag\'esimo%
4243     \or octog\'esimo%
4244     \or nonag\'esimo%
4245   \fi
4246 }%
4247 \global\let\@@tenthstringportuges\@@tenthstringportuges

  Teens:
4248 \newcommand*\@@teenthstringportuges[1]{%
4249   \@tenthstring{1}%
4250   \ifnum#1>0\relax
4251     -\@unitthstring{#1}%
4252   \fi
4253 }%
4254 \global\let\@@teenthstringportuges\@@teenthstringportuges

  Hundreds:
4255 \newcommand*\@@hundredthstringportuges[1]{%
4256   \ifcase#1\relax
4257     \or cent\'esimo%
4258     \or ducent\'esimo%
4259     \or trecent\'esimo%
4260     \or quadringent\'esimo%
4261     \or q\'uingent\'esimo%
4262     \or seiscent\'esimo%
4263     \or setingent\'esimo%
4264     \or octingent\'esimo%
4265     \or nongent\'esimo%
4266   \fi
4267 }%
4268 \global\let\@@hundredthstringportuges\@@hundredthstringportuges

  Units (feminine):
4269 \newcommand*\@@unitthstringFportuges[1]{%
4270   \ifcase#1\relax
4271     zero%
4272     \or primeira%
4273     \or segunda%
4274     \or terceira%
4275     \or quarta%
4276     \or quinta%
4277     \or sexta%
4278     \or s\'etima%

```

```

4279 \or oitava%
4280 \or nona%
4281 \fi
4282 }%
4283 \global\let\@@unitthstringFportuges\@@unitthstringFportuges

```

Tens (feminine):

```

4284 \newcommand*\@@tenthstringFportuges[1]{%
4285 \ifcase#1\relax
4286 \or d\`ecima%
4287 \or vig\`esima%
4288 \or trig\`esima%
4289 \or quadrag\`esima%
4290 \or q\`uinquag\`esima%
4291 \or sexag\`esima%
4292 \or setuag\`esima%
4293 \or octog\`esima%
4294 \or nonag\`esima%
4295 \fi
4296 }%
4297 \global\let\@@tenthstringFportuges\@@tenthstringFportuges

```

Hundreds (feminine):

```

4298 \newcommand*\@@hundredthstringFportuges[1]{%
4299 \ifcase#1\relax
4300 \or cent\`esima%
4301 \or ducent\`esima%
4302 \or trecent\`esima%
4303 \or quadringent\`esima%
4304 \or q\`uingent\`esima%
4305 \or seiscent\`esima%
4306 \or setingent\`esima%
4307 \or octingent\`esima%
4308 \or nongent\`esima%
4309 \fi
4310 }%
4311 \global\let\@@hundredthstringFportuges\@@hundredthstringFportuges

```

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

```

4312 \newcommand*\@@Unitthstringportuges[1]{%
4313 \ifcase#1\relax
4314 Zero%
4315 \or Primeiro%
4316 \or Segundo%
4317 \or Terceiro%
4318 \or Quarto%
4319 \or Quinto%
4320 \or Sexto%
4321 \or S\`etimo%
4322 \or Oitavo%
4323 \or Nono%

```

```

4324 \fi
4325 }%
4326 \global\let\@@Unitthstringportuges\@@Unitthstringportuges

Tens:
4327 \newcommand*\@@Tenthstringportuges[1]{%
4328 \ifcase#1\relax
4329 \or D\'ecimo%
4330 \or Vig\'esimo%
4331 \or Trig\'esimo%
4332 \or Quadrag\'esimo%
4333 \or Q\'uinquag\'esimo%
4334 \or Sexag\'esimo%
4335 \or Setuag\'esimo%
4336 \or Octog\'esimo%
4337 \or Nonag\'esimo%
4338 \fi
4339 }%
4340 \global\let\@@Tenthstringportuges\@@Tenthstringportuges

Hundreds:
4341 \newcommand*\@@Hundredthstringportuges[1]{%
4342 \ifcase#1\relax
4343 \or Cent\'esimo%
4344 \or Ducent\'esimo%
4345 \or Trecent\'esimo%
4346 \or Quadringent\'esimo%
4347 \or Q\'uingent\'esimo%
4348 \or Seiscent\'esimo%
4349 \or Setingent\'esimo%
4350 \or Octingent\'esimo%
4351 \or Nongent\'esimo%
4352 \fi
4353 }%
4354 \global\let\@@Hundredthstringportuges\@@Hundredthstringportuges

As above, but feminine. Units:
4355 \newcommand*\@@UnitthstringFportuges[1]{%
4356 \ifcase#1\relax
4357 Zera%
4358 \or Primeira%
4359 \or Segunda%
4360 \or Terceira%
4361 \or Quarta%
4362 \or Quinta%
4363 \or Sexta%
4364 \or S\'etima%
4365 \or Oitava%
4366 \or Nona%
4367 \fi
4368 }%

```

4369 \global\let\@@UnitthstringFportuges\@@UnitthstringFportuges

Tens (feminine);

4370 \newcommand*\@@TenthstringFportuges[1]{%

4371 \ifcase#1\relax

4372 \or D\'esima%

4373 \or Vig\'esima%

4374 \or Trig\'esima%

4375 \or Quadrag\'esima%

4376 \or Q\'uinquag\'esima%

4377 \or Sexag\'esima%

4378 \or Setuag\'esima%

4379 \or Octog\'esima%

4380 \or Nonag\'esima%

4381 \fi

4382 }%

4383 \global\let\@@TenthstringFportuges\@@TenthstringFportuges

Hundreds (feminine):

4384 \newcommand*\@@HundredthstringFportuges[1]{%

4385 \ifcase#1\relax

4386 \or Cent\'esima%

4387 \or Ducent\'esima%

4388 \or Trecent\'esima%

4389 \or Quadringent\'esima%

4390 \or Q\'uingent\'esima%

4391 \or Seiscent\'esima%

4392 \or Setingent\'esima%

4393 \or Octingent\'esima%

4394 \or Nongent\'esima%

4395 \fi

4396 }%

4397 \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.)

4398 \newcommand*\@@numberstringportuges[2]{%

4399 \ifnum#1>99999\relax

4400 \PackageError{fmtcount}{Out of range}%

4401 {This macro only works for values less than 100000}%

4402 \else

4403 \ifnum#1<0\relax

4404 \PackageError{fmtcount}{Negative numbers not permitted}%

4405 {This macro does not work for negative numbers, however

4406 you can try typing "minus" first, and then pass the modulus of

4407 this number}%

4408 \fi

4409 \fi

```

4410 \def#2{}%
4411 \@strctr=#1\relax \divide\@strctr by 1000\relax
4412 \ifnum\@strctr>9\relax

  #1 is greater or equal to 10000

4413 \divide\@strctr by 10\relax
4414 \ifnum\@strctr>1\relax
4415 \let\@fc@numstr#2\relax
4416 \protected@edef#2{\@fc@numstr\@tenstring{\@strctr}}%
4417 \@strctr=#1 \divide\@strctr by 1000\relax
4418 \@FCmodulo{\@strctr}{10}%
4419 \ifnum\@strctr>0
4420 \ifnum\@strctr=1\relax
4421 \let\@fc@numstr#2\relax
4422 \protected@edef#2{\@fc@numstr\ \@andname}%
4423 \fi
4424 \let\@fc@numstr#2\relax
4425 \protected@edef#2{\@fc@numstr\ \@unitstring{\@strctr}}%
4426 \fi
4427 \else
4428 \@strctr=#1\relax
4429 \divide\@strctr by 1000\relax
4430 \@FCmodulo{\@strctr}{10}%
4431 \let\@fc@numstr#2\relax
4432 \protected@edef#2{\@fc@numstr\@teenstring{\@strctr}}%
4433 \fi
4434 \let\@fc@numstr#2\relax
4435 \protected@edef#2{\@fc@numstr\ \@thousand}%
4436 \else
4437 \ifnum\@strctr>0\relax
4438 \ifnum\@strctr>1\relax
4439 \let\@fc@numstr#2\relax
4440 \protected@edef#2{\@fc@numstr\@unitstring{\@strctr}\ }%
4441 \fi
4442 \let\@fc@numstr#2\relax
4443 \protected@edef#2{\@fc@numstr\@thousand}%
4444 \fi
4445 \fi
4446 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
4447 \divide\@strctr by 100\relax
4448 \ifnum\@strctr>0\relax
4449 \ifnum#1>1000 \relax
4450 \let\@fc@numstr#2\relax
4451 \protected@edef#2{\@fc@numstr\ }%
4452 \fi
4453 \@tmpstrctr=#1\relax
4454 \@FCmodulo{\@tmpstrctr}{1000}%
4455 \let\@fc@numstr#2\relax
4456 \ifnum\@tmpstrctr=100\relax
4457 \protected@edef#2{\@fc@numstr\@tenstring{10}}%

```

```

4458 \else
4459 \protected@edef#2{\@fc@numstr\@hundredstring{\@strctr}}%
4460 \fi%
4461 \fi
4462 \@strctr=#1\relax \@FCmodulo{\@strctr}{100}%
4463 \ifnum#1>100\relax
4464 \ifnum\@strctr>0\relax
4465 \let\@fc@numstr#2\relax
4466 \protected@edef#2{\@fc@numstr\ \@andname\ }%
4467 \fi
4468 \fi
4469 \ifnum\@strctr>19\relax
4470 \divide\@strctr by 10\relax
4471 \let\@fc@numstr#2\relax
4472 \protected@edef#2{\@fc@numstr\@tenstring{\@strctr}}%
4473 \@strctr=#1\relax \@FCmodulo{\@strctr}{10}%
4474 \ifnum\@strctr>0
4475 \ifnum\@strctr=1\relax
4476 \let\@fc@numstr#2\relax
4477 \protected@edef#2{\@fc@numstr\ \@andname}%
4478 \else
4479 \ifnum#1>100\relax
4480 \let\@fc@numstr#2\relax
4481 \protected@edef#2{\@fc@numstr\ \@andname}%
4482 \fi
4483 \fi
4484 \let\@fc@numstr#2\relax
4485 \protected@edef#2{\@fc@numstr\ \@unitstring{\@strctr}}%
4486 \fi
4487 \else
4488 \ifnum\@strctr<10\relax
4489 \ifnum\@strctr=0\relax
4490 \ifnum#1<100\relax
4491 \let\@fc@numstr#2\relax
4492 \protected@edef#2{\@fc@numstr\@unitstring{\@strctr}}%
4493 \fi
4494 \else %(>0,<10)
4495 \let\@fc@numstr#2\relax
4496 \protected@edef#2{\@fc@numstr\@unitstring{\@strctr}}%
4497 \fi
4498 \else%>10
4499 \@FCmodulo{\@strctr}{10}%
4500 \let\@fc@numstr#2\relax
4501 \protected@edef#2{\@fc@numstr\@teenstring{\@strctr}}%
4502 \fi
4503 \fi
4504 }%
4505 \global\let\@numberstringportuges\@numberstringportuges

```

As above, but for ordinals.

```

4506 \newcommand*{\@@ordinalstringportuges[2]}{%
4507 \@strctr=#1\relax
4508 \ifnum#1>99999
4509 \PackageError{fmtcount}{Out of range}%
4510 {This macro only works for values less than 100000}%
4511 \else
4512 \ifnum#1<0
4513 \PackageError{fmtcount}{Negative numbers not permitted}%
4514 {This macro does not work for negative numbers, however
4515 you can try typing "minus" first, and then pass the modulus of
4516 this number}%
4517 \else
4518 \def#2{}%
4519 \ifnum\@strctr>999\relax
4520 \divide\@strctr by 1000\relax
4521 \ifnum\@strctr>1\relax
4522 \ifnum\@strctr>9\relax
4523 \@tmpstrctr=\@strctr
4524 \ifnum\@strctr<20
4525 \@FCmodulo{\@tmpstrctr}{10}%
4526 \let\@@fc@ordstr#2\relax
4527 \protected@edef#2{\@@fc@ordstr\@teenthstring{\@tmpstrctr}}%
4528 \else
4529 \divide\@tmpstrctr by 10\relax
4530 \let\@@fc@ordstr#2\relax
4531 \protected@edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
4532 \@tmpstrctr=\@strctr
4533 \@FCmodulo{\@tmpstrctr}{10}%
4534 \ifnum\@tmpstrctr>0\relax
4535 \let\@@fc@ordstr#2\relax
4536 \protected@edef#2{\@@fc@ordstr\@unitthstring{\@tmpstrctr}}%
4537 \fi
4538 \fi
4539 \else
4540 \let\@@fc@ordstr#2\relax
4541 \protected@edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
4542 \fi
4543 \fi
4544 \let\@@fc@ordstr#2\relax
4545 \protected@edef#2{\@@fc@ordstr\@thousandth}%
4546 \fi
4547 \@strctr=#1\relax
4548 \@FCmodulo{\@strctr}{1000}%
4549 \ifnum\@strctr>99\relax
4550 \@tmpstrctr=\@strctr
4551 \divide\@tmpstrctr by 100\relax
4552 \ifnum#1>1000\relax
4553 \let\@@fc@ordstr#2\relax
4554 \protected@edef#2{\@@fc@ordstr-}%

```

```

4555 \fi
4556 \let\@@fc@ordstr#2\relax
4557 \protected@edef#2{\@@fc@ordstr\@hundredthstring{\@tmpstrctr}}%
4558 \fi
4559 \@FCmodulo{\@strctr}{100}%
4560 \ifnum#1>99\relax
4561 \ifnum\@strctr>0\relax
4562 \let\@@fc@ordstr#2\relax
4563 \protected@edef#2{\@@fc@ordstr-}%
4564 \fi
4565 \fi
4566 \ifnum\@strctr>9\relax
4567 \@tmpstrctr=\@strctr
4568 \divide\@tmpstrctr by 10\relax
4569 \let\@@fc@ordstr#2\relax
4570 \protected@edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
4571 \@tmpstrctr=\@strctr
4572 \@FCmodulo{\@tmpstrctr}{10}%
4573 \ifnum\@tmpstrctr>0\relax
4574 \let\@@fc@ordstr#2\relax
4575 \protected@edef#2{\@@fc@ordstr-\@unitthstring{\@tmpstrctr}}%
4576 \fi
4577 \else
4578 \ifnum\@strctr=0\relax
4579 \ifnum#1=0\relax
4580 \let\@@fc@ordstr#2\relax
4581 \protected@edef#2{\@@fc@ordstr\@unitstring{0}}%
4582 \fi
4583 \else
4584 \let\@@fc@ordstr#2\relax
4585 \protected@edef#2{\@@fc@ordstr\@unitthstring{\@strctr}}%
4586 \fi
4587 \fi
4588 \fi
4589 \fi
4590 }%
4591 \global\let\@@ordinalstringportuges\@@ordinalstringportuges

```

9.4.13 fc-portuguese.def

```
4592 \ProvidesFCLanguage{portuguese}[2014/06/09]%
```

Load fc-portuges.def if not already loaded

```
4593 \FCloadlang{portuges}%
```

Set portuguese to be equivalent to portuges.

```

4594 \global\let\@ordinalMportuguese=\@ordinalMportuges
4595 \global\let\@ordinalFportuguese=\@ordinalFportuges
4596 \global\let\@ordinalNportuguese=\@ordinalNportuges
4597 \global\let\@numberstringMportuguese=\@numberstringMportuges

```

```

4598 \global\let \@numberstringFportuguese=\@numberstringFportuges
4599 \global\let \@numberstringNportuguese=\@numberstringNportuges
4600 \global\let \@NumberstringMportuguese=\@NumberstringMportuges
4601 \global\let \@NumberstringFportuguese=\@NumberstringFportuges
4602 \global\let \@NumberstringNportuguese=\@NumberstringNportuges
4603 \global\let \@ordinalstringMportuguese=\@ordinalstringMportuges
4604 \global\let \@ordinalstringFportuguese=\@ordinalstringFportuges
4605 \global\let \@ordinalstringNportuguese=\@ordinalstringNportuges
4606 \global\let \@OrdinalstringMportuguese=\@OrdinalstringMportuges
4607 \global\let \@OrdinalstringFportuguese=\@OrdinalstringFportuges
4608 \global\let \@OrdinalstringNportuguese=\@OrdinalstringNportuges

```

9.4.14 fc-spanish.def

Spanish definitions

```
4609 \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:

```

4610 \newcommand* \@ordinalMspanish[2]{%
4611   \edef#2{\number#1\relax\noexpand\fmtord{o}}%
4612 }%
4613 \global\let \@ordinalMspanish \@ordinalMspanish

```

Feminine:

```

4614 \newcommand* \@ordinalFspanish[2]{%
4615   \edef#2{\number#1\relax\noexpand\fmtord{a}}%
4616 }%
4617 \global\let \@ordinalFspanish \@ordinalFspanish

```

Make neuter same as masculine:

```
4618 \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):

```

4619 \newcommand* @@unitstringspanish[1]{%
4620   \ifcase#1\relax
4621     cero%
4622     \or uno%
4623     \or dos%
4624     \or tres%
4625     \or cuatro%
4626     \or cinco%
4627     \or seis%
4628     \or siete%
4629     \or ocho%
4630     \or nueve%
4631   \fi
4632 }%

```

4633 \global\let\@@unitstringspanish\@@unitstringspanish

Feminine:

4634 \newcommand*\@@unitstringFspanish[1]{%

4635 \ifcase#1\relax

4636 cera%

4637 \or una%

4638 \or dos%

4639 \or tres%

4640 \or cuatro%

4641 \or cinco%

4642 \or seis%

4643 \or siete%

4644 \or ocho%

4645 \or nueve%

4646 \fi

4647 }%

4648 \global\let\@@unitstringFspanish\@@unitstringFspanish

Tens (argument must go from 1 to 10):

4649 \newcommand*\@@tenstringspanish[1]{%

4650 \ifcase#1\relax

4651 \or diez%

4652 \or veinte%

4653 \or treinta%

4654 \or cuarenta%

4655 \or cincuenta%

4656 \or sesenta%

4657 \or setenta%

4658 \or ochenta%

4659 \or noventa%

4660 \or cien%

4661 \fi

4662 }%

4663 \global\let\@@tenstringspanish\@@tenstringspanish

Teens:

4664 \newcommand*\@@teenstringspanish[1]{%

4665 \ifcase#1\relax

4666 diez%

4667 \or once%

4668 \or doce%

4669 \or trece%

4670 \or catorce%

4671 \or quince%

4672 \or diecis\'eis%

4673 \or diecisiete%

4674 \or dieciocho%

4675 \or diecinueve%

4676 \fi

4677 }%

4678 \global\let\@@teenstringspanish\@@teenstringspanish

Twenties:

4679 \newcommand*\@@twentystringspanish[1]{%

4680 \ifcase#1\relax

4681 veinte%

4682 \or veintiuno%

4683 \or veintid\'os%

4684 \or veintitr\'es%

4685 \or veinticuatro%

4686 \or veinticinco%

4687 \or veintis\'eis%

4688 \or veintisiete%

4689 \or veintiocho%

4690 \or veintinueve%

4691 \fi

4692 }%

4693 \global\let\@@twentystringspanish\@@twentystringspanish

Feminine form:

4694 \newcommand*\@@twentystringFspanish[1]{%

4695 \ifcase#1\relax

4696 veinte%

4697 \or veintiuna%

4698 \or veintid\'os%

4699 \or veintitr\'es%

4700 \or veinticuatro%

4701 \or veinticinco%

4702 \or veintis\'eis%

4703 \or veintisiete%

4704 \or veintiocho%

4705 \or veintinueve%

4706 \fi

4707 }%

4708 \global\let\@@twentystringFspanish\@@twentystringFspanish

Hundreds:

4709 \newcommand*\@@hundredstringspanish[1]{%

4710 \ifcase#1\relax

4711 \or ciento%

4712 \or doscientos%

4713 \or trescientos%

4714 \or cuatrocientos%

4715 \or quinientos%

4716 \or seiscientos%

4717 \or setecientos%

4718 \or ochocientos%

4719 \or novecientos%

4720 \fi

4721 }%

4722 \global\let\@@hundredstringspanish\@@hundredstringspanish

Feminine form:

```
4723 \newcommand*{\@@hundredstringFspanish[1]}{%
4724   \ifcase#1\relax
4725     \or cienta%
4726     \or doscientas%
4727     \or trescientas%
4728     \or cuatrocientas%
4729     \or quinientas%
4730     \or seiscientas%
4731     \or setecientas%
4732     \or ochocientas%
4733     \or novecientas%
4734   \fi
4735 }%
4736 \global\let\@@hundredstringFspanish\@@hundredstringFspanish
```

As above, but with initial letter uppercase:

```
4737 \newcommand*{\@@Unitstringspanish[1]}{%
4738   \ifcase#1\relax
4739     Cero%
4740     \or Uno%
4741     \or Dos%
4742     \or Tres%
4743     \or Cuatro%
4744     \or Cinco%
4745     \or Seis%
4746     \or Siete%
4747     \or Ocho%
4748     \or Nueve%
4749   \fi
4750 }%
4751 \global\let\@@Unitstringspanish\@@Unitstringspanish
```

Feminine form:

```
4752 \newcommand*{\@@UnitstringFspanish[1]}{%
4753   \ifcase#1\relax
4754     Cera%
4755     \or Una%
4756     \or Dos%
4757     \or Tres%
4758     \or Cuatro%
4759     \or Cinco%
4760     \or Seis%
4761     \or Siete%
4762     \or Ocho%
4763     \or Nueve%
4764   \fi
4765 }%
4766 \global\let\@@UnitstringFspanish\@@UnitstringFspanish
```

Tens:

```

4767%\changes{2.0}{2012-06-18}{fixed spelling mistake (correction
4768 %provided by Fernando Maldonado)}
4769 \newcommand*\@@Tenstringspanish[1]{%
4770   \ifcase#1\relax
4771     \or Diez%
4772     \or Veinte%
4773     \or Treinta%
4774     \or Cuarenta%
4775     \or Cincuenta%
4776     \or Sesenta%
4777     \or Setenta%
4778     \or Ochenta%
4779     \or Noventa%
4780     \or Cien%
4781   \fi
4782 }%
4783 \global\let\@@Tenstringspanish\@@Tenstringspanish

```

Teens:

```

4784 \newcommand*\@@Teenstringspanish[1]{%
4785   \ifcase#1\relax
4786     Diez%
4787     \or Once%
4788     \or Doce%
4789     \or Trece%
4790     \or Catorce%
4791     \or Quince%
4792     \or Diecis\'eis%
4793     \or Diecisiete%
4794     \or Dieciocho%
4795     \or Diecinueve%
4796   \fi
4797 }%
4798 \global\let\@@Teenstringspanish\@@Teenstringspanish

```

Twenties:

```

4799 \newcommand*\@@Twentystingspanish[1]{%
4800   \ifcase#1\relax
4801     Veinte%
4802     \or Veintiuno%
4803     \or Veintid\'os%
4804     \or Veintitr\'es%
4805     \or Veinticuatro%
4806     \or Veinticinco%
4807     \or Veintis\'eis%
4808     \or Veintisiete%
4809     \or Veintiocho%
4810     \or Veintinueve%
4811   \fi
4812 }%

```

4813 \global\let\@@Twentystringspanish\@@Twentystringspanish

Feminine form:

4814 \newcommand*\@@TwentystringFspanish[1]{%

4815 \ifcase#1\relax

4816 Veinte%

4817 \or Veintiuna%

4818 \or Veintid\'os%

4819 \or Veintitr\'es%

4820 \or Veinticuatro%

4821 \or Veinticinco%

4822 \or Veintis\'eis%

4823 \or Veintisiete%

4824 \or Veintiocho%

4825 \or Veintinueve%

4826 \fi

4827 }%

4828 \global\let\@@TwentystringFspanish\@@TwentystringFspanish

Hundreds:

4829 \newcommand*\@@Hundredstringspanish[1]{%

4830 \ifcase#1\relax

4831 \or Ciento%

4832 \or Doscientos%

4833 \or Trescientos%

4834 \or Cuatrocientos%

4835 \or Quinientos%

4836 \or Seiscientos%

4837 \or Setecientos%

4838 \or Ochocientos%

4839 \or Novecientos%

4840 \fi

4841 }%

4842 \global\let\@@Hundredstringspanish\@@Hundredstringspanish

Feminine form:

4843 \newcommand*\@@HundredstringFspanish[1]{%

4844 \ifcase#1\relax

4845 \or Cienta%

4846 \or Doscientas%

4847 \or Trescientas%

4848 \or Cuatrocientas%

4849 \or Quinientas%

4850 \or Seiscientas%

4851 \or Setecientas%

4852 \or Ochocientas%

4853 \or Novecientas%

4854 \fi

4855 }%

4856 \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.)

```
4857 \DeclareRobustCommand{\@numberstringMspanish}[2]{%
4858   \let\@unitstring=\@unitstringspanish
4859   \let\@teenstring=\@teenstringspanish
4860   \let\@tenstring=\@tenstringspanish
4861   \let\@twentystring=\@twentystringspanish
4862   \let\@hundredstring=\@hundredstringspanish
4863   \def\@hundred{cien}\def\@thousand{mil}%
4864   \def\@andname{y}%
4865   \@numberstringspanish{#1}{#2}%
4866 }%
4867 \global\let\@numberstringMspanish\@numberstringMspanish
```

Feminine form:

```
4868 \DeclareRobustCommand{\@numberstringFspanish}[2]{%
4869   \let\@unitstring=\@unitstringFspanish
4870   \let\@teenstring=\@teenstringspanish
4871   \let\@tenstring=\@tenstringspanish
4872   \let\@twentystring=\@twentystringFspanish
4873   \let\@hundredstring=\@hundredstringFspanish
4874   \def\@hundred{cien}\def\@thousand{mil}%
4875   \def\@andname{b}%
4876   \@numberstringspanish{#1}{#2}%
4877 }%
4878 \global\let\@numberstringFspanish\@numberstringFspanish
```

Make neuter same as masculine:

```
4879 \global\let\@numberstringNspanish\@numberstringMspanish
```

As above, but initial letters in upper case:

```
4880 \DeclareRobustCommand{\@NumberstringMspanish}[2]{%
4881   \let\@unitstring=\@Unitstringspanish
4882   \let\@teenstring=\@Teenstringspanish
4883   \let\@tenstring=\@Tenstringspanish
4884   \let\@twentystring=\@Twentystringspanish
4885   \let\@hundredstring=\@Hundredstringspanish
4886   \def\@andname{y}%
4887   \def\@hundred{Cien}\def\@thousand{Mil}%
4888   \@numberstringspanish{#1}{#2}%
4889 }%
4890 \global\let\@NumberstringMspanish\@NumberstringMspanish
```

Feminine form:

```
4891 \DeclareRobustCommand{\@NumberstringFspanish}[2]{%
4892   \let\@unitstring=\@UnitstringFspanish
4893   \let\@teenstring=\@Teenstringspanish
4894   \let\@tenstring=\@Tenstringspanish
```

```

4895 \let\@twentystring=\@TwentystringFspanish
4896 \let\@hundredstring=\@HundredstringFspanish
4897 \def\@andname{b}%
4898 \def\@hundred{Cien}\def\@thousand{Mil}%
4899 \@numberstringspanish{#1}{#2}%
4900 }%
4901 \global\let\@NumberstringFspanish\@NumberstringFspanish

  Make neuter same as masculine:
4902 \global\let\@NumberstringNspanish\@NumberstringMspanish

  As above, but for ordinals.
4903 \DeclareRobustCommand{\@ordinalstringMspanish}[2]{%
4904 \let\@unitthstring=\@unitthstringspanish
4905 \let\@unitstring=\@unitstringspanish
4906 \let\@teenthstring=\@teenthstringspanish
4907 \let\@tenthstring=\@tenthstringspanish
4908 \let\@hundredthstring=\@hundredthstringspanish
4909 \def\@thousandth{mil\'esimo}%
4910 \@ordinalstringspanish{#1}{#2}%
4911 }%
4912 \global\let\@ordinalstringMspanish\@ordinalstringMspanish

  Feminine form:
4913 \DeclareRobustCommand{\@ordinalstringFspanish}[2]{%
4914 \let\@unitthstring=\@unitthstringFspanish
4915 \let\@unitstring=\@unitstringFspanish
4916 \let\@teenthstring=\@teenthstringFspanish
4917 \let\@tenthstring=\@tenthstringFspanish
4918 \let\@hundredthstring=\@hundredthstringFspanish
4919 \def\@thousandth{mil\'esima}%
4920 \@ordinalstringspanish{#1}{#2}%
4921 }%
4922 \global\let\@ordinalstringFspanish\@ordinalstringFspanish

  Make neuter same as masculine:
4923 \global\let\@ordinalstringNspanish\@ordinalstringMspanish

  As above, but with initial letters in upper case.
4924 \DeclareRobustCommand{\@OrdinalstringMspanish}[2]{%
4925 \let\@unitthstring=\@Unitthstringspanish
4926 \let\@unitstring=\@Unitstringspanish
4927 \let\@teenthstring=\@Teenthstringspanish
4928 \let\@tenthstring=\@Tenthstringspanish
4929 \let\@hundredthstring=\@Hundredthstringspanish
4930 \def\@thousandth{Mil\'esimo}%
4931 \@ordinalstringspanish{#1}{#2}%
4932 }
4933 \global\let\@OrdinalstringMspanish\@OrdinalstringMspanish

  Feminine form:
4934 \DeclareRobustCommand{\@OrdinalstringFspanish}[2]{%

```

```

4935 \let\@unitthstring=\@UnitthstringFspanish
4936 \let\@unitstring=\@UnitstringFspanish
4937 \let\@teenthstring=\@TeenthstringFspanish
4938 \let\@tenthstring=\@TenthstringFspanish
4939 \let\@hundredthstring=\@HundredthstringFspanish
4940 \def\@thousandth{Mil\'esima}%
4941 \@ordinalstringspanish{#1}{#2}%
4942 }%
4943 \global\let\@OrdinalstringFspanish\@OrdinalstringFspanish

```

Make neuter same as masculine:

```

4944 \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:

```

4945 \newcommand*\@unitthstringspanish[1]{%
4946 \ifcase#1\relax
4947   cero%
4948   \or primero%
4949   \or segundo%
4950   \or tercero%
4951   \or cuarto%
4952   \or quinto%
4953   \or sexto%
4954   \or s\'eptimo%
4955   \or octavo%
4956   \or noveno%
4957 \fi
4958 }%
4959 \global\let\@unitthstringspanish\@unitthstringspanish

```

Tens:

```

4960 \newcommand*\@tenthstringspanish[1]{%
4961 \ifcase#1\relax
4962   \or d\'ecimo%
4963   \or vig\'esimo%
4964   \or trig\'esimo%
4965   \or cuadrag\'esimo%
4966   \or quincuag\'esimo%
4967   \or sexag\'esimo%
4968   \or septuag\'esimo%
4969   \or octog\'esimo%
4970   \or nonag\'esimo%
4971 \fi
4972 }%
4973 \global\let\@tenthstringspanish\@tenthstringspanish

```

Teens:

```

4974 \newcommand*\@teenthstringspanish[1]{%
4975 \ifcase#1\relax
4976   d\'ecimo%

```

```

4977 \or und\'ecimo%
4978 \or duod\'ecimo%
4979 \or decimotercero%
4980 \or decimocuarto%
4981 \or decimoquinto%
4982 \or decimosexto%
4983 \or decimos\'eptimo%
4984 \or decimoctavo%
4985 \or decimonoveno%
4986 \fi
4987 }%
4988 \global\let\@@teenthstringspanish\@@teenthstringspanish

  Hundreds:
4989 \newcommand*\@@hundredthstringspanish[1]{%
4990 \ifcase#1\relax
4991 \or cent\'esimo%
4992 \or ducent\'esimo%
4993 \or tricent\'esimo%
4994 \or cuadringent\'esimo%
4995 \or quingent\'esimo%
4996 \or sexcent\'esimo%
4997 \or septing\'esimo%
4998 \or octingent\'esimo%
4999 \or noningent\'esimo%
5000 \fi
5001 }%
5002 \global\let\@@hundredthstringspanish\@@hundredthstringspanish

  Units (feminine):
5003 \newcommand*\@@unitthstringFspanish[1]{%
5004 \ifcase#1\relax
5005 cera%
5006 \or primera%
5007 \or segunda%
5008 \or tercera%
5009 \or cuarta%
5010 \or quinta%
5011 \or sexta%
5012 \or s\'eptima%
5013 \or octava%
5014 \or novena%
5015 \fi
5016 }%
5017 \global\let\@@unitthstringFspanish\@@unitthstringFspanish

  Tens (feminine):
5018 \newcommand*\@@tenthstringFspanish[1]{%
5019 \ifcase#1\relax
5020 \or d\'ecima%
5021 \or vig\'esima%

```

```

5022 \or trig\'esima%
5023 \or cuadrag\'esima%
5024 \or quincuag\'esima%
5025 \or sexag\'esima%
5026 \or septuag\'esima%
5027 \or octog\'esima%
5028 \or nonag\'esima%
5029 \fi
5030 }%
5031 \global\let\@@tenthstringFspanish\@@tenthstringFspanish
    Teens (feminine)
5032 \newcommand*\@@teenthstringFspanish[1]{%
5033 \ifcase#1\relax
5034 d\'ecima%
5035 \or und\'ecima%
5036 \or duod\'ecima%
5037 \or decimotercera%
5038 \or decimocuarta%
5039 \or decimoquinta%
5040 \or decimosexta%
5041 \or decimos\'eptima%
5042 \or decimoctava%
5043 \or decimonovena%
5044 \fi
5045 }%
5046 \global\let\@@teenthstringFspanish\@@teenthstringFspanish
    Hundreds (feminine)
5047 \newcommand*\@@hundredthstringFspanish[1]{%
5048 \ifcase#1\relax
5049 \or cent\'esima%
5050 \or ducent\'esima%
5051 \or tricent\'esima%
5052 \or cuadringent\'esima%
5053 \or quingent\'esima%
5054 \or sexcent\'esima%
5055 \or septing\'esima%
5056 \or octingent\'esima%
5057 \or noningent\'esima%
5058 \fi
5059 }%
5060 \global\let\@@hundredthstringFspanish\@@hundredthstringFspanish
    As above, but with initial letters in upper case
5061 \newcommand*\@@Unitthstringspanish[1]{%
5062 \ifcase#1\relax
5063 Cero%
5064 \or Primero%
5065 \or Segundo%
5066 \or Tercero%

```

```

5067 \or Cuarto%
5068 \or Quinto%
5069 \or Sexto%
5070 \or S\'eptimo%
5071 \or Octavo%
5072 \or Noveno%
5073 \fi
5074 }%
5075 \global\let\@@Unitthstringspanish\@@Unitthstringspanish

```

Tens:

```

5076 \newcommand*\@@Tenthstringspanish[1]{%
5077 \ifcase#1\relax
5078 \or D\'ecimo%
5079 \or Vig\'esimo%
5080 \or Trig\'esimo%
5081 \or Cuadrag\'esimo%
5082 \or Quincuag\'esimo%
5083 \or Sexag\'esimo%
5084 \or Septuag\'esimo%
5085 \or Octog\'esimo%
5086 \or Nonag\'esimo%
5087 \fi
5088 }%
5089 \global\let\@@Tenthstringspanish\@@Tenthstringspanish

```

Teens:

```

5090 \newcommand*\@@Teenthstringspanish[1]{%
5091 \ifcase#1\relax
5092 D\'ecimo%
5093 \or Und\'ecimo%
5094 \or Duod\'ecimo%
5095 \or Decimotercero%
5096 \or Decimocuarto%
5097 \or Decimoquinto%
5098 \or Decimosexto%
5099 \or Decimos\'eptimo%
5100 \or Decimoctavo%
5101 \or Decimonoveno%
5102 \fi
5103 }%
5104 \global\let\@@Teenthstringspanish\@@Teenthstringspanish

```

Hundreds

```

5105 \newcommand*\@@Hundredthstringspanish[1]{%
5106 \ifcase#1\relax
5107 \or Cent\'esimo%
5108 \or Ducent\'esimo%
5109 \or Tricent\'esimo%
5110 \or Cuadringent\'esimo%
5111 \or Quingent\'esimo%

```

```

5112 \or Sexcent\'esimo%
5113 \or Septing\'esimo%
5114 \or Octingent\'esimo%
5115 \or Noningent\'esimo%
5116 \fi
5117 }%
5118 \global\let\@@Hundredthstringspanish\@@Hundredthstringspanish

```

As above, but feminine.

```

5119 \newcommand*\@@UnitthstringFspanish[1]{%
5120 \ifcase#1\relax
5121 Cera%
5122 \or Primera%
5123 \or Segunda%
5124 \or Tercera%
5125 \or Cuarta%
5126 \or Quinta%
5127 \or Sexta%
5128 \or S\'eptima%
5129 \or Octava%
5130 \or Novena%
5131 \fi
5132 }%
5133 \global\let\@@UnitthstringFspanish\@@UnitthstringFspanish

```

Tens (feminine)

```

5134 \newcommand*\@@TenthstringFspanish[1]{%
5135 \ifcase#1\relax
5136 \or D\'ecima%
5137 \or Vig\'esima%
5138 \or Trig\'esima%
5139 \or Cuadrag\'esima%
5140 \or Quincuag\'esima%
5141 \or Sexag\'esima%
5142 \or Septuag\'esima%
5143 \or Octog\'esima%
5144 \or Nonag\'esima%
5145 \fi
5146 }%
5147 \global\let\@@TenthstringFspanish\@@TenthstringFspanish

```

Teens (feminine):

```

5148 \newcommand*\@@TeenthstringFspanish[1]{%
5149 \ifcase#1\relax
5150 D\'ecima%
5151 \or Und\'ecima%
5152 \or Duod\'ecima%
5153 \or Decimotercera%
5154 \or Decimocuarta%
5155 \or Decimoquinta%
5156 \or Decimosexta%

```

```

5157 \or Decimos\'eptima%
5158 \or Decimoctava%
5159 \or Decimonovena%
5160 \fi
5161 }%
5162 \global\let\@@TeenthstringFspanish\@@TeenthstringFspanish

```

Hundreds (feminine):

```

5163 \newcommand*\@@HundredthstringFspanish[1]{%
5164 \ifcase#1\relax
5165 \or Cent\'esima%
5166 \or Ducent\'esima%
5167 \or Tricent\'esima%
5168 \or Cuadringent\'esima%
5169 \or Quingent\'esima%
5170 \or Sexcent\'esima%
5171 \or Septing\'esima%
5172 \or Octingent\'esima%
5173 \or Noningent\'esima%
5174 \fi
5175 }%
5176 \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.)

```

5177 \newcommand*\@@numberstringspanish[2]{%
5178 \ifnum#1>99999
5179 \PackageError{fmtcount}{Out of range}%
5180 {This macro only works for values less than 100000}%
5181 \else
5182 \ifnum#1<0
5183 \PackageError{fmtcount}{Negative numbers not permitted}%
5184 {This macro does not work for negative numbers, however
5185 you can try typing "minus" first, and then pass the modulus of
5186 this number}%
5187 \fi
5188 \fi
5189 \def#2{}%
5190 \@strctr=#1\relax \divide\@strctr by 1000\relax
5191 \ifnum\@strctr>9
5192 \divide\@strctr by 10
5193 \ifnum\@strctr>1
5194 \let\@@fc@numstr#2\relax
5195 \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
5196 \@strctr=#1 \divide\@strctr by 1000\relax
5197 \@FCmodulo{\@strctr}{10}%

```

```

5198 \ifnum \@strctr>0\relax
5199 \let\@@fc@numstr#2\relax
5200 \edef#2{\@@fc@numstr\ \@andname\ \@unitstring{\@strctr}}%
5201 \fi
5202 \else
5203 \@strctr=#1\relax
5204 \divide\@strctr by 1000\relax
5205 \@FCmodulo{\@strctr}{10}%
5206 \let\@@fc@numstr#2\relax
5207 \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
5208 \fi
5209 \let\@@fc@numstr#2\relax
5210 \edef#2{\@@fc@numstr\ \@thousand}%
5211 \else
5212 \ifnum \@strctr>0\relax
5213 \ifnum \@strctr>1\relax
5214 \let\@@fc@numstr#2\relax
5215 \edef#2{\@@fc@numstr\@unitstring{\@strctr}\ }%
5216 \fi
5217 \let\@@fc@numstr#2\relax
5218 \edef#2{\@@fc@numstr\@thousand}%
5219 \fi
5220 \fi
5221 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
5222 \divide\@strctr by 100\relax
5223 \ifnum \@strctr>0\relax
5224 \ifnum#1>1000\relax
5225 \let\@@fc@numstr#2\relax
5226 \edef#2{\@@fc@numstr\ }%
5227 \fi
5228 \@tmpstrctr=#1\relax
5229 \@FCmodulo{\@tmpstrctr}{1000}%
5230 \ifnum \@tmpstrctr=100\relax
5231 \let\@@fc@numstr#2\relax
5232 \edef#2{\@@fc@numstr\@tenstring{10}}%
5233 \else
5234 \let\@@fc@numstr#2\relax
5235 \edef#2{\@@fc@numstr\@hundredstring{\@strctr}}%
5236 \fi
5237 \fi
5238 \@strctr=#1\relax \@FCmodulo{\@strctr}{100}%
5239 \ifnum#1>100\relax
5240 \ifnum \@strctr>0\relax
5241 \let\@@fc@numstr#2\relax
5242 \edef#2{\@@fc@numstr\ }%
5243 \fi
5244 \fi
5245 \ifnum \@strctr>29\relax
5246 \divide\@strctr by 10\relax

```

```

5247 \let\@fc@numstr#2\relax
5248 \edef#2{\@fc@numstr\@tenstring{\@strctr}}%
5249 \@strctr=#1\relax \@FCmodulo{\@strctr}{10}%
5250 \ifnum\@strctr>0\relax
5251 \let\@fc@numstr#2\relax
5252 \edef#2{\@fc@numstr\@andname\@unitstring{\@strctr}}%
5253 \fi
5254 \else
5255 \ifnum\@strctr<10\relax
5256 \ifnum\@strctr=0\relax
5257 \ifnum#1<100\relax
5258 \let\@fc@numstr#2\relax
5259 \edef#2{\@fc@numstr\@unitstring{\@strctr}}%
5260 \fi
5261 \else
5262 \let\@fc@numstr#2\relax
5263 \edef#2{\@fc@numstr\@unitstring{\@strctr}}%
5264 \fi
5265 \else
5266 \ifnum\@strctr>19\relax
5267 \@FCmodulo{\@strctr}{10}%
5268 \let\@fc@numstr#2\relax
5269 \edef#2{\@fc@numstr\@twentystring{\@strctr}}%
5270 \else
5271 \@FCmodulo{\@strctr}{10}%
5272 \let\@fc@numstr#2\relax
5273 \edef#2{\@fc@numstr\@teenstring{\@strctr}}%
5274 \fi
5275 \fi
5276 \fi
5277 }%
5278 \global\let\@numberstringspanish\@numberstringspanish

```

As above, but for ordinals

```

5279 \newcommand*\@ordinalstringspanish[2]{%
5280 \@strctr=#1\relax
5281 \ifnum#1>99999
5282 \PackageError{fmtcount}{Out of range}%
5283 {This macro only works for values less than 100000}%
5284 \else
5285 \ifnum#1<0
5286 \PackageError{fmtcount}{Negative numbers not permitted}%
5287 {This macro does not work for negative numbers, however
5288 you can try typing "minus" first, and then pass the modulus of
5289 this number}%
5290 \else
5291 \def#2{}%
5292 \ifnum\@strctr>999\relax
5293 \divide\@strctr by 1000\relax
5294 \ifnum\@strctr>1\relax

```

```

5295 \ifnum \@strctr>9\relax
5296 \@tmpstrctr=\@strctr
5297 \ifnum \@strctr<20
5298 \@FCmodulo{\@tmpstrctr}{10}%
5299 \let\@@fc@ordstr#2\relax
5300 \edef#2{\@@fc@ordstr\@teenthstring{\@tmpstrctr}}%
5301 \else
5302 \divide\@tmpstrctr by 10\relax
5303 \let\@@fc@ordstr#2\relax
5304 \edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
5305 \@tmpstrctr=\@strctr
5306 \@FCmodulo{\@tmpstrctr}{10}%
5307 \ifnum \@tmpstrctr>0\relax
5308 \let\@@fc@ordstr#2\relax
5309 \edef#2{\@@fc@ordstr\@unitthstring{\@tmpstrctr}}%
5310 \fi
5311 \fi
5312 \else
5313 \let\@@fc@ordstr#2\relax
5314 \edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
5315 \fi
5316 \fi
5317 \let\@@fc@ordstr#2\relax
5318 \edef#2{\@@fc@ordstr\@thousandth}%
5319 \fi
5320 \@strctr=#1\relax
5321 \@FCmodulo{\@strctr}{1000}%
5322 \ifnum \@strctr>99\relax
5323 \@tmpstrctr=\@strctr
5324 \divide\@tmpstrctr by 100\relax
5325 \ifnum#1>1000\relax
5326 \let\@@fc@ordstr#2\relax
5327 \edef#2{\@@fc@ordstr\ }%
5328 \fi
5329 \let\@@fc@ordstr#2\relax
5330 \edef#2{\@@fc@ordstr\@hundredthstring{\@tmpstrctr}}%
5331 \fi
5332 \@FCmodulo{\@strctr}{100}%
5333 \ifnum#1>99\relax
5334 \ifnum \@strctr>0\relax
5335 \let\@@fc@ordstr#2\relax
5336 \edef#2{\@@fc@ordstr\ }%
5337 \fi
5338 \fi
5339 \ifnum \@strctr>19\relax
5340 \@tmpstrctr=\@strctr
5341 \divide\@tmpstrctr by 10\relax
5342 \let\@@fc@ordstr#2\relax
5343 \edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%

```

```

5344 \@tmpstrctr=\@strctr
5345 \@FCmodulo{\@tmpstrctr}{10}%
5346 \ifnum \@tmpstrctr>0\relax
5347   \let\@fc@ordstr#2\relax
5348   \edef#2{\@fc@ordstr\ \@unitthstring{\@tmpstrctr}}%
5349 \fi
5350 \else
5351   \ifnum \@strctr>9\relax
5352     \@FCmodulo{\@strctr}{10}%
5353     \let\@fc@ordstr#2\relax
5354     \edef#2{\@fc@ordstr\@teenthstring{\@strctr}}%
5355   \else
5356     \ifnum \@strctr=0\relax
5357       \ifnum#1=0\relax
5358         \let\@fc@ordstr#2\relax
5359         \edef#2{\@fc@ordstr\@unitstring{0}}%
5360       \fi
5361     \else
5362       \let\@fc@ordstr#2\relax
5363       \edef#2{\@fc@ordstr\@unitthstring{\@strctr}}%
5364     \fi
5365   \fi
5366 \fi
5367 \fi
5368 \fi
5369 }%
5370 \global\let\@ordinalstringspanish\@ordinalstringspanish

```

9.4.15 fc-UKenglish.def

English definitions

```
5371 \ProvidesFCLanguage{UKenglish}[2013/08/17]%
```

Loaded fc-english.def if not already loaded

```
5372 \FCloadlang{english}%
```

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

```

5373 \global\let\@ordinalMUKenglish\@ordinalMenglish
5374 \global\let\@ordinalFUKenglish\@ordinalMenglish
5375 \global\let\@ordinalNUKenglish\@ordinalMenglish
5376 \global\let\@numberstringMUKenglish\@numberstringMenglish
5377 \global\let\@numberstringFUKenglish\@numberstringMenglish
5378 \global\let\@numberstringNUKenglish\@numberstringMenglish
5379 \global\let\@NumberstringMUKenglish\@NumberstringMenglish
5380 \global\let\@NumberstringFUKenglish\@NumberstringMenglish
5381 \global\let\@NumberstringNUKenglish\@NumberstringMenglish
5382 \global\let\@ordinalstringMUKenglish\@ordinalstringMenglish
5383 \global\let\@ordinalstringFUKenglish\@ordinalstringMenglish
5384 \global\let\@ordinalstringNUKenglish\@ordinalstringMenglish
5385 \global\let\@OrdinalstringMUKenglish\@OrdinalstringMenglish

```

```
5386 \global\let\@OrdinalstringFUKenglish\@OrdinalstringMenglish
5387 \global\let\@OrdinalstringNUKenglish\@OrdinalstringMenglish
```

9.4.16 fc-USenglish.def

US English definitions

```
5388 \ProvidesFCLanguage{USenglish}[2013/08/17]%
```

Loaded fc-english.def if not already loaded

```
5389 \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.)

```
5390 \global\let\@ordinalMUSenglish\@ordinalMenglish
5391 \global\let\@ordinalFUSenglish\@ordinalMenglish
5392 \global\let\@ordinalNUSenglish\@ordinalMenglish
5393 \global\let\@numberstringMUSenglish\@numberstringMenglish
5394 \global\let\@numberstringFUSenglish\@numberstringMenglish
5395 \global\let\@numberstringNUSenglish\@numberstringMenglish
5396 \global\let\@NumberstringMUSenglish\@NumberstringMenglish
5397 \global\let\@NumberstringFUSenglish\@NumberstringMenglish
5398 \global\let\@NumberstringNUSenglish\@NumberstringMenglish
5399 \global\let\@ordinalstringMUSenglish\@ordinalstringMenglish
5400 \global\let\@ordinalstringFUSenglish\@ordinalstringMenglish
5401 \global\let\@ordinalstringNUSenglish\@ordinalstringMenglish
5402 \global\let\@OrdinalstringMUSenglish\@OrdinalstringMenglish
5403 \global\let\@OrdinalstringFUSenglish\@OrdinalstringMenglish
5404 \global\let\@OrdinalstringNUSenglish\@OrdinalstringMenglish
```