fmtcount.sty: Displaying the Values of LATEX Counters

Nicola L.C. Talbot
www.dickimaw-books.com

Vincent Belaïche

2013-08-17 (version 2.03)

Contents

1 Introduction

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

2 Available Commands

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

`\ordinal`

```
\ordinal{⟨counter⟩}[⟨gender⟩]
```

This will print the value of a LaTeX counter ⟨*counter*⟩ as an ordinal, where the macro

`\fmtord`

```
\fmtord{⟨text⟩}
```

is used to format the st, nd, rd, th bit. By default the ordinal is formatted as a superscript, if the package option level is used, it is level with the text. For example, if the current section is 3, then `\ordinal{section}` will produce the output: 3$^{\text{rd}}$. 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 incompatiblity, if you want to use the fmtcount package with the memoir class you should use

`\FCordinal`

```
\FCordinal
```

to access fmtcount's version of `\ordinal`, and use `\ordinal` to use memoir's version of that command.

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

The commands below only work for numbers in the range 0 to 99999.

`\ordinalnum`

`\ordinalnum{⟨n⟩}[⟨gender⟩]`

This is like `\ordinal` but takes an actual number rather than a counter as the argument. For example: `\ordinalnum{3}` will produce: 3rd.

`\numberstring`

`\numberstring{⟨counter⟩}[⟨gender⟩]`

This will print the value of ⟨*counter*⟩ as text. E.g. `\numberstring{section}` will produce: three. The optional argument is the same as that for `\ordinal`.

`\Numberstring`

`\Numberstring{⟨counter⟩}[⟨gender⟩]`

This does the same as `\numberstring`, but with initial letters in uppercase. For example, `\Numberstring{section}` will produce: Three.

`\NUMBERstring`

`\NUMBERstring{⟨counter⟩}[⟨gender⟩]`

This does the same as `\numberstring`, but converts the string to upper case. Note that `\MakeUppercase{\NUMBERstring{⟨counter⟩}}` doesn't work, due to the way that `\MakeUppercase` expands its argument[1].

`\numberstringnum`

`\numberstringnum{⟨n⟩}[⟨gender⟩]`

`\Numberstringnum`

`\Numberstringnum{⟨n⟩}[⟨gender⟩]`

`\NUMBERstringnum`

`\NUMBERstringnum{⟨n⟩}[⟨gender⟩]`

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

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

`\ordinalstring`

```
\ordinalstring{⟨counter⟩}[⟨gender⟩]
```

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

`\Ordinalstring`

```
\Ordinalstring{⟨counter⟩}[⟨gender⟩]
```

This does the same as `\ordinalstring`, but with initial letters in uppercase. For example, `\Ordinalstring{section}` will produce: Third.

`\ORDINALstring`

```
\ORDINALstring{⟨counter⟩}[⟨gender⟩]
```

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

`\ordinalstringnum`

```
\ordinalstringnum{⟨n⟩}[⟨gender⟩]
```

`\Ordinalstringnum`

```
\Ordinalstringnum{⟨n⟩}[⟨gender⟩]
```

`\ORDINALstringnum`

```
\ORDINALstringnum{⟨n⟩}[⟨gender⟩]
```

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

As from version 1.09, textual representations can be stored for later use. This overcomes the problems encountered when you attempt to use one of the above commands in `\edef`.

Each of the following commands takes a label as the first argument, the other arguments are as the analogous commands above. These commands do not display anything, but store the textual representation. This can later be retrieved using

`\FMCuse`

```
\FMCuse{⟨label⟩}
```

Note: with `\storeordinal` and `\storeordinalnum`, the only bit that doesn't get expanded is `\fmtord`. So, for example, `\storeordinalnum{mylabel}{3}` will be stored as `3\relax \fmtord{rd}`.

\storeordinal

\storeordinal{⟨*label*⟩}{⟨*counter*⟩}[⟨*gender*⟩]

\storeordinalstring

\storeordinalstring{⟨*label*⟩}{⟨*counter*⟩}[⟨*gender*⟩]

\storeOrdinalstring

\storeOrdinalstring{⟨*label*⟩}{⟨*counter*⟩}[⟨*gender*⟩]

\storeORDINALstring

\storeORDINALstring{⟨*label*⟩}{⟨*counter*⟩}[⟨*gender*⟩]

\storenumberstring

\storenumberstring{⟨*label*⟩}{⟨*counter*⟩}[⟨*gender*⟩]

\storeNumberstring

\storeNumberstring{⟨*label*⟩}{⟨*counter*⟩}[⟨*gender*⟩]

\storeNUMBERstring

\storeNUMBERstring{⟨*label*⟩}{⟨*counter*⟩}[⟨*gender*⟩]

\storeordinalnum

\storeordinalnum{⟨*label*⟩}{⟨*number*⟩}[⟨*gender*⟩]

oreordinalstringnum

\storeordinalstring{⟨*label*⟩}{⟨*number*⟩}[⟨*gender*⟩]

oreOrdinalstringnum

\storeOrdinalstringnum{⟨*label*⟩}{⟨*number*⟩}[⟨*gender*⟩]

oreORDINALstringnum

\storeORDINALstringnum{⟨*label*⟩}{⟨*number*⟩}[⟨*gender*⟩]

torenumberstringnum

\storenumberstring{⟨*label*⟩}{⟨*number*⟩}[⟨*gender*⟩]

toreNumberstringnum

\storeNumberstring{⟨*label*⟩}{⟨*number*⟩}[⟨*gender*⟩]

toreNUMBERstringnum

\storeNUMBERstring{⟨*label*⟩}{⟨*number*⟩}[⟨*gender*⟩]

`\binary`

```
\binary{⟨counter⟩}
```

This will print the value of ⟨*counter*⟩ as a binary number. E.g. `\binary{section}` will produce: 11. The declaration

`\padzeroes`

```
\padzeroes[⟨n⟩]
```

will ensure numbers are written to ⟨*n*⟩ digits, padding with zeroes if necessary. E.g. `\padzeroes[8]\binary{section}` will produce: 00000011. The default value for ⟨*n*⟩ is 17.

`\binarynum`

```
\binary{⟨n⟩}
```

This is like `\binary` but takes an actual number rather than a counter as the argument. For example: `\binarynum{5}` will produce: 101.

The octal commands only work for values in the range 0 to 32768.

`\octal`

```
\octal{⟨counter⟩}
```

This will print the value of ⟨*counter*⟩ as an octal number. For example, if you have a counter called, say `mycounter`, and you set the value to 125, then `\octal{mycounter}` will produce: 177. Again, the number will be padded with zeroes if necessary, depending on whether `\padzeroes` has been used.

`\octalnum`

```
\octalnum{⟨n⟩}
```

This is like `\octal` but takes an actual number rather than a counter as the argument. For example: `\octalnum{125}` will produce: 177.

`\hexadecimal`

```
\hexadecimal{⟨counter⟩}
```

This will print the value of ⟨*counter*⟩ as a hexadecimal number. Going back to the counter used in the previous example, `\hexadecimal{mycounter}` will produce: 7d. Again, the number will be padded with zeroes if necessary, depending on whether `\padzeroes` has been used.

`\Hexadecimal`

```
\Hexadecimal{⟨counter⟩}
```

This does the same thing, but uses uppercase characters, e.g. `\Hexadecimal{mycounter}` will produce: 7D.

`\hexadecimalnum`

```
\hexadecimalnum{⟨n⟩}
```

`\Hexadecimalnum`

```
\Hexadecimalnum{⟨n⟩}
```

These are like `\hexadecimal` and `\Hexadecimal` but take an actual number rather than a counter as the argument. For example: `\hexadecimalnum{125}` will produce: 7d, and `\Hexadecimalnum{125}` will produce: 7D.

`\decimal`

```
\decimal{⟨counter⟩}
```

This is similar to `\arabic` but the number can be padded with zeroes depending on whether `\padzeroes` has been used. For example: `\padzeroes[8]\decimal{section}` will produce: 00000005.

`\decimalnum`

```
\decimalnum{⟨n⟩}
```

This is like `\decimal` but takes an actual number rather than a counter as the argument. For example: `\padzeroes[8]\decimalnum{5}` will produce: 00000005.

`\aaalph`

```
\aaalph{⟨counter⟩}
```

This will print the value of ⟨*counter*⟩ as: a b ... z aa bb ... zz etc. For example, `\aaalpha{mycounter}` will produce: uuuuu if `mycounter` is set to 125.

`\AAAlph`

```
\AAAlph{⟨counter⟩}
```

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

`\aaalphnum`

```
\aaalphnum{⟨n⟩}
```

`\AAAlphnum`

```
\AAAlphnum{⟨n⟩}
```

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

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

`\abalph`

```
\abalph{⟨counter⟩}
```

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

`\ABAlph`

```
\ABAlph{⟨counter⟩}
```

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

\abalphnum

```
\abalphnum{⟨n⟩}
```

\ABAlphnum

```
\ABAlphnum{⟨n⟩}
```

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

3 Package Options

The following options can be passed to this package:

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

These can also be set using the command:

fmtcountsetoptions

```
\fmtcountsetoptions{fmtord=⟨type⟩}
```

where ⟨*type*⟩ is either `level` or `raise`.

4 Multilingual Support

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

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

\FCloadlang

```
\FCloadlang{⟨dialect⟩}
```

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

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

[2]Thanks to K. H. Fricke for supplying the information.
[3]Thanks to Edoardo Pasca for supplying the information.

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

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

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

4.1 Options for French

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

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

`\fmtcountsetoptions`

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

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

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

`dialect`

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

`french`

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

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

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

france	soixante-dix pour 70, quatre-vingts pour 80, et quate-vingts-dix pour 90,
belgian	septante pour 70, quatre-vingts pour 80, et nonante pour 90,
swiss	septante pour 70, huitante[4] pour 80, et nonante pour 90

[4]voir Octante et huitante sur le site d'Alain Lassine

Il est à noter que la variante `belgian` est parfaitement correcte pour les francophones français[5], 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 2eme (par defaut)

`vingt plural`

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

`cent plural`

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

`mil plural`

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

`n-illion plural`

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

`n-illiard plural`

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

`all plural`

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

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

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

[5] je précise que l'auteur de ces lignes est français

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

- un autre cas demandant quelque configurabilité est celui de « mil » et « mille ». Pour rappel « mille » est le pluriel irrégulier de « mil », mais l'alternance mil/mille est rare, voire pédante, car aujourd'hui « mille » n'est utilisé que comme un mot invariable, en effet le sort des pluriels étrangers est systématiquement de finir par disparaître comme par exemple « scénarii » aujourd'hui supplanté par « scénarios ». Pour continuer à pouvoir écrire « mil », il aurait fallu former le pluriel comme « mils », ce qui n'est pas l'usage. Certaines personnes utilisent toutefois encore « mil » dans les dates, par exemple « mil neuf cent quatre-vingt quatre » au lieu de « mille neuf cent quatre-vingt quatre »,

- finalement les règles du français quoique bien définies ne sont pas très cohérentes et il est donc inévitable qu'un jour ou l'autre on 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 ***globalement*** 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 ***localement*** en dernière position, où “localement” siginifie 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 globablement*** en dernière position, où “localement” et *globablement* on la même siginification 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’infère être en vigueur pour les nombres de la forme $\langle n\rangle$illion et $\langle n\rangle$illiard lorsque le nombre a une valeur ordinale, mais à dire vrai pour des nombres aussi grands, par exemple « deux millions », je pense qu’il n’est tout simplement pas d’usage de dire « l’exemplaire deux million(s ?) » pour « le deux millionième exemplaire ».

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

⟨*x*⟩ dans "⟨*x*⟩ `plural`"	`traditional`	`reformed`	`traditional o`	`reformed o`
`vingt`	multiple l-last		multiple lng-last	
`cent`				
`mil`	always			
`n-illion`	multiple		multiple ng-last	
`n-illiard`				

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

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

`dash or space`

`\fmtcountsetoptions{french={dash or space=`⟨*dash or space*⟩`}}`

Avant la réforme de l'orthographe de 1990, on ne met des traits d'union qu'entre les dizaines et les unités, et encore sauf quand le nombre n considéré est tel que $n \mod 10 = 1$, 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 pise 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 $n > 0$,
infty	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⟩]
```

`tinnumeralstringnum`

```
\latinnumeralstringnum{⟨number⟩}[⟨prefix options⟩]
```

5 Configuration File fmtcount.cfg

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

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

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

and if `fmtcount.cfg` has the line:

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

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

6 LaTeX2HTML style

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

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

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

  This, I agree, is an unpleasant cludge.

7 Acknowledgements

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

8 Troubleshooting

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

9 The Code

9.1 fcnumparser.sty

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

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

```
\def\fc@counter@parser#1{%
  \expandafter\fc@number@parser\expandafter{\the#1.}%
}
\newcount\fc@digit@counter

\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`.

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

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

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

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

No comma either, so fractional part is set empty.

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

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

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

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

```
      \def\@tempb##1.\fc@end{\def\fc@fractional@part{##1}}%
      \expandafter\@tempb\@tempa
  \fi
}
```

\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@`$\langle n\rangle$, and macros `\fc@min@weight` and `\fc@max@weight` are set to the bounds for $\langle n\rangle$.

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

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

```
  \let\@tempa\@empty
  \def\@tempb##1##2\fc@nil{%
    \def\@tempc{##1}%
    \ifx\@tempc\space
    \else
      \expandafter\def\expandafter\@tempa\expandafter{\@tempa ##1}%
    \fi
    \def\@tempc{##2}%
    \ifx\@tempc\@empty
      \expandafter\@gobble
    \else
      \expandafter\@tempb
    \fi
    ##2\fc@nil
  }%
  \@tempb#1\fc@nil
```

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

```
\def\@tempb##1##2\fc@nil{\def\fc@sign{##1}\def\fc@number{##2}}%
\expandafter\@tempb\@tempa\fc@nil
\expandafter\if\fc@sign+%
  \def\fc@sign@case{1}%
\else
  \expandafter\if\fc@sign-%
    \def\fc@sign@case{2}%
  \else
    \def\fc@sign{}%
    \def\fc@sign@case{0}%
    \let\fc@number\@tempa
  \fi
\fi
\ifx\fc@number\@empty
  \PackageError{fcnumparser}{Invalid number}{Number must contain at least one non blank
    character after sign}%
\fi
```

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

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

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

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

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

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

```
\fc@digit@counter -\fc@digit@counter
\advance\fc@digit@counter by \fc@unit@weight
```

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

Now we loop for i = `\fc@unit@weight` to `\fc@max@weight` in order to copy all the digits from `\fc@digit@`⟨i + offset⟩ to `\fc@digit@`⟨i⟩. First we compute offset into `\@tempi`.

```
{%
  \count0 \fc@unit@weight\relax
  \count1 \fc@max@weight\relax
  \advance\count0 by -\count1 %
  \advance\count0 by -1 %
  \def\@tempa##1{\def\@tempb{\def\@tempi{##1}}}%
  \expandafter\@tempa\expandafter{\the\count0}%
  \expandafter
}\@tempb
```

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

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

Here is the loop body:

```
      {%
        \count0 \@tempi
        \advance\count0 by \fc@digit@counter
        \expandafter\def\expandafter\@tempd\expandafter{\csname fc@digit@\the\count0\endcs
        \expandafter\def\expandafter\@tempe\expandafter{\csname fc@digit@\the\fc@digit@cou
        \def\@tempa####1####2{\def\@tempb{\let####1####2}}%
        \expandafter\expandafter\expandafter\@tempa\expandafter\@tempe\@tempd
        \expandafter
      }\@tempb
      \advance\fc@digit@counter by 1 %
   \fi
   \next
}%
\let\next\@templ
\@templ
```

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

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

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

```
\@ifundefined{fc@parse@integer@digits}{}{%
  \PackageError{fcnumparser}{Duplicate definition}{Redefinition of
    macro ‘fc@parse@integer@digits’}}
\def\fc@parse@integer@digits#1#2\fc@nil{%
  \def\@tempa{#1}%
  \ifx\@tempa\fc@end@
     \def\next##1\fc@nil{}%
  \else
  \let\next\fc@parse@integer@digits
  \advance\fc@digit@counter by -1
  \expandafter\def\csname fc@digit@\the\fc@digit@counter\endcsname{#1}%
  \fi
  \next#2\fc@nil
}


\newcommand*{\fc@unit@weight}{0}

```

Now we have macros to read a few digits from the `\fc@digit@`⟨*n*⟩ array and form a correspoding 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.

```
\@ifundefined{fc@read@unit}{}{%
  \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.

```
\def\fc@read@unit#1#2{%
  \ifnum#2>\fc@max@weight
     #1=0\relax
  \else
     \ifnum#2<\fc@min@weight
        #1=0\relax
     \else
         {%
           \edef\@tempa{\number#2}%
           \count0=\@tempa
           \edef\@tempa{\csname fc@digit@\the\count0\endcsname}%
           \def\@tempb##1{\def\@tempa{#1=##1\relax}}%
           \expandafter\@tempb\expandafter{\@tempa}%
           \expandafter
         }\@tempa
     \fi
  \fi
}
```

\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.

```
\@ifundefined{fc@read@hundred}{}{%
  \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

```
\def\fc@read@hundred#1#2{%
  {%
    \fc@read@unit{\count0}{#2}%
    \def\@tempa##1{\fc@read@unit{\count1}{##1}}%
    \count2=#2%
    \advance\count2 by 1 %
    \expandafter\@tempa{\the\count2}%
    \multiply\count1 by 10 %
    \advance\count1 by \count0 %
    \def\@tempa##1{\def\@tempb{#1=##1\relax}}
    \expandafter\@tempa\expandafter{\the\count1}%
    \expandafter
  }\@tempb
}
```

\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.

```
\@ifundefined{fc@read@thousand}{}{%
  \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro
    ‘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

```
\def\fc@read@thousand#1#2{%
  {%
    \fc@read@unit{\count0}{#2}%
    \def\@tempa##1{\fc@read@hundred{\count1}{##1}}%
    \count2=#2%
    \advance\count2 by 1 %
    \expandafter\@tempa{\the\count2}%
    \multiply\count1 by 10 %
    \advance\count1 by \count0 %
    \def\@tempa##1{\def\@tempb{#1=##1\relax}}
    \expandafter\@tempa\expandafter{\the\count1}%
    \expandafter
  }\@tempb
}
```

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

First we check that the macro is not yet defined.

```
\@ifundefined{fc@read@myriad}{}{%
  \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro
    ‘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

```
\def\fc@read@myriad#1#2{%
  {%
    \fc@read@hundred{\count0}{#2}%
    \def\@tempa##1{\fc@read@hundred{\count1}{##1}}%
    \count2=#2
    \advance\count2 by 2
    \expandafter\@tempa{\the\count2}%
    \multiply\count1 by 100 %
    \advance\count1 by \count0 %
    \def\@tempa##1{\def\@tempb{#1=##1\relax}}%
    \expandafter\@tempa\expandafter{\the\count1}%
    \expandafter
  }\@tempb
}
```

`\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.

```
\@ifundefined{fc@check@nonzeros}{}{%
  \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro
    ‘fc@check@nonzeros’}}
```

Arguments as follows:

#1 input number: minimum unit unit weight at which start to search the non-zeros

#2 input number: maximum unit weight at which end to seach the non-zeros

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

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

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

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

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

Then we do the real job

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

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

```
  \def\@tempd##1{\def\@tempa{\def#3{##1}}}%
  \expandafter\@tempd\expandafter{\@tempc}%
  \expandafter
 }\@tempa
}
```

`\fc` @@check@nonzeros@inner Macro `\fc@@check@nonzeros@inner` Check wehther 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 unit weight at which start to search the non-zeros; *output* macro may have been redefined
`\@tempb`	input/output macro: *input* maximum unit weight at which end to seach the non-zeros; *output* macro may have been redefined
`\@tempc`	ouput macro: 0 if all-zeros, 1 if at least one zero is found
`\count0`	output counter: weight + 1 of the first found non zero starting from minimum weight.

```
\def\fc@@check@nonzeros@inner{%
  \ifnum\count0<\fc@min@weight
     \count0=\fc@min@weight\relax
  \fi
  \ifnum\count1>\fc@max@weight\relax
     \count1=\fc@max@weight
  \fi
  \count2\count0 %
  \advance\count2 by 1 %
  \ifnum\count0>\count1 %
    \PackageError{fcnumparser}{Unexpected arguments}{Number in argument 2 of macro
      ‘fc@check@nonzeros’ must be at least equal to number in argument 1}%
  \else
    \fc@@check@nonzeros@inner@loopbody
    \ifnum\@tempc>0 %
      \ifnum\@tempc<9 %
        \ifnum\count0>\count1 %
        \else
          \let\@tempd\@tempc
          \fc@@check@nonzeros@inner@loopbody
          \ifnum\@tempc=0 %
            \let\@tempc\@tempd
          \else
            \def\@tempc{9}%
          \fi
        \fi
      \fi
    \fi
  \fi
}
\def\fc@@check@nonzeros@inner@loopbody{%
  % \@tempc <-  digit of weight \count0
  \expandafter\let\expandafter\@tempc\csname fc@digit@\the\count0\endcsname
  \advance\count0 by 1 %
  \ifnum\@tempc=0 %
     \ifnum\count0>\count1 %
       \let\next\relax
     \else
       \let\next\fc@@check@nonzeros@inner@loopbody
     \fi
  \else
     \ifnum\count0>\count2 %
       \def\@tempc{9}%
     \fi
     \let\next\relax
  \fi
  \next
}
```

`\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.

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

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

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

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

```
    \count0=\fc@unit@weight\space
    \ifnum\count0<\fc@min@weight\space
      \count0=\fc@min@weight\space
    \fi
```

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

```
    \def\@templ{%
      \ifnum\csname fc@digit@\the\count0\endcsname=0 %
        \advance\count0 by 1 %
        \ifnum\count0>\fc@max@weight\space
          \let\next\relax
        \fi
      \else
        \let\next\relax
      \fi
      \next
    }%
    \let\next\@templ
    \@templ
```

Now propagate result after closing bracket into counter #1.

```
     \toks0{#1}%
     \edef\@tempa{\the\toks0=\the\count0}%
     \expandafter
  }\@tempa\space
}
```

\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

```
\@ifundefined{fc@get@last@word}{}{\PackageError{fcnumparser}{Duplicate definition}{Redefinit
    of macro ‘fc@get@last@word’}}%
\def\fc@get@last@word#1#2#3{%
  {%
```

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.

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

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

```
    \def\@tempa##1\fc@case##2\fc@end{%
      \toks2{##1}%
      \def\@tempb{##2}%
      \toks3{##2}%
    }%
```

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

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

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

```
    \def\@templ{%
      \expandafter\@tempa\the\toks1 \fc@end
      \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.
```
        \let\next\relax
      \else
```
\@tempb is not empty, first we use
```
          \expandafter\expandafter\expandafter\@tempt
          \expandafter\expandafter\expandafter{%
            \expandafter\the\expandafter\@tempt
            \expandafter\fc@case\the\toks2}%
          \toks1\toks3 %
      \fi
      \next
    }%
    \let\next\@templ
    \@templ
    \edef\@tempa{\def\noexpand#2{\the\toks0}\def\noexpand#3{\the\toks2}}%
    \expandafter
  }\@tempa
}
```
\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
```
\@ifundefined{fc@get@last@letter}{}{\PackageError{fcnumparser}{Duplicate definition}{Redefi
    of macro ‘fc@get@last@letter’}}%
\def\fc@get@last@letter#1#2#3{%
  {%
```
First copy input to local \toks1. What we are going to 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.
```
    \toks1{#1}%
    \toks0{}%
    \toksdef\@tempt0 %
```
We define \@tempa in order to pop the first letter from the remaining of word.
```
    \def\@tempa##1##2\fc@nil{%
      \toks2{##1}%
      \toks3{##2}%
      \def\@tempb{##2}%
    }%
```
Now we define \@templ to do the loop by terminal recursion.
```
    \def\@templ{%
      \expandafter\@tempa\the\toks1 \fc@nil
      \ifx\@tempb\@empty
```
Stop loop, as \toks1 has been detected to be one single letter.

```
        \let\next\relax
      \else
```

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

```
         \expandafter\expandafter\expandafter\@tempt
         \expandafter\expandafter\expandafter{%
           \expandafter\the\expandafter\@tempt
           \the\toks2}%
```

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

```
        \toks1\toks3 %
      \fi
      \next
    }%
```

Here run the loop.

```
    \let\next\@templ
    \next
```

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

```
    \edef\@tempa{\def\noexpand#2{\the\toks0}\def\noexpand#3{\the\toks1}}%
    \expandafter
  }\@tempa
}%
```

9.2 fcprefix.sty

Pseudo-latin prefixes.

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{fcprefix}[2012/09/28]
\RequirePackage{ifthen}
\RequirePackage{keyval}
\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`'.

```
\define@key{fcprefix}{use duode and unde}[below20]{%
  \ifthenelse{\equal{#1}{below20}}{%
    \def\fc@duodeandunde{2}%
  }{%
    \ifthenelse{\equal{#1}{never}}{%
        \def\fc@duodeandunde{0}%
    }{%
      \PackageError{fcprefix}{Unexpected option}{%
        Option 'use duode and unde' expects 'below 20' or 'never' }%
    }%
  }%
}
```

Default is '`below 20`' like in French.

```
\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.

```
\define@key{fcprefix}{numeral u in duo}[false]{%
  \ifthenelse{\equal{#1}{false}}{%
    \let\fc@u@in@duo\@empty
  }{%
    \ifthenelse{\equal{#1}{true}}{%
      \def\fc@u@in@duo{u}%
    }{%
      \PackageError{fcprefix}{Unexpected option}{%
        Option ‘numeral u in duo’ expects ‘true’ or ‘false’ }%
    }%
  }%
}
```

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.

```
\define@key{fcprefix}{e accute}[false]{%
  \ifthenelse{\equal{#1}{false}}{%
    \let\fc@prefix@eaccute\@firstofone
  }{%
    \ifthenelse{\equal{#1}{true}}{%
      \let\fc@prefix@eaccute\'%
    }{%
      \PackageError{fcprefix}{Unexpected option}{%
        Option ‘e accute’ expects ‘true’ or ‘false’ }%
    }%
  }%
}
```

Default is to set accute accent like in French.

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

Option '`power of millia`' tells how millia is raise to power n. It expects value:

`recursive`	for which millia squared is noted as 'milliamillia'
`arabic`	for which millia squared is noted as 'millia^2'
`prefix`	for which millia squared is noted as 'bismillia'

```
\define@key{fcprefix}{power of millia}[prefix]{%
  \ifthenelse{\equal{#1}{prefix}}{%
      \let\fc@power@of@millia@init\@gobbletwo
      \let\fc@power@of@millia\fc@@prefix@millia
  }{%
    \ifthenelse{\equal{#1}{arabic}}{%
      \let\fc@power@of@millia@init\@gobbletwo
      \let\fc@power@of@millia\fc@@arabic@millia
    }{%
      \ifthenelse{\equal{#1}{recursive}}{%
        \let\fc@power@of@millia@init\fc@@recurse@millia@init
        \let\fc@power@of@millia\fc@@recurse@millia
      }{%
        \PackageError{fcprefix}{Unexpected option}{%
          Option ‘power of millia’ expects ‘recursive’, ‘arabic’, or ‘prefix’ }%
      }%
    }%
  }%
}
```

Arguments as follows:

#1	output macro
#2	number with current weight w

```
\def\fc@@recurse@millia#1#2{%
  \let\@tempp#1%
  \edef#1{millia\@tempp}%
}
```

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

#1	output macro
#2	number with current weight w

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

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

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

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

```
    \count0=0 %
    \let\@tempa\@empty
    \loop
       \ifnum\count0<\@tempb
         \advance\count0 by 1 %
         \expandafter\def
           \expandafter\@tempa\expandafter{\@tempa millia}%
    \repeat
```

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

```
    \edef\@tempb{\def\noexpand#1{\@tempa}}%
    \expandafter
  }\@tempb
}
```

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

#1	output macro
#2	number with current weight w

```
\def\fc@@arabic@millia#1#2{%
  \ifnnum#2=0 %
    \let#1\@empty
  \else
    \edef#1{millia\^{}\the#2}%
  \fi
}
```

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

#1 output macro

#2 number with current weight w

```
\def\fc@@prefix@millia#1#2{%
  \fc@@latin@numeral@pefix{#2}{#1}%
}
```

Default value of option 'power of millia' is 'prefix':

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

`\fc` @@latin@cardinal@pefix Compute a cardinal prefix for n-illion, like 1 ⇒ 'm', 2 ⇒ 'bi', 3 ⇒ '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.

```
\@ifundefined{fc@@latin@cardinal@pefix}{}{%
  \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro 'fc@@latin@cardinal@p
```

Arguments as follows:

#1 input number to be formated

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

```
\def\fc@@latin@cardinal@pefix#1#2{%
  {%
```

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

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

Now parse number from expanded input.

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

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

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

`\@tempm` will hold the `millia^`$^{n\div3}$

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

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

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

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

```
      \advance\count2 by 1 %
      \fc@read@unit{\count5}{\count2}%
      \advance\count2 by 1 %
```
If the 3 considered digits $d_2d_1d_0$ are not all zero, then set `\@tempt` to '`t`' for the first time this event is met.
```
      \edef\@tempn{%
        \ifnum\count3=0\else 1\fi
        \ifnum\count4=0\else 1\fi
        \ifnum\count5=0\else 1\fi
      }%
      \ifx\@tempn\@empty\else
        \let\@tempt\@tempu
        \let\@tempu\@empty
      \fi
```
Now process the current group $d_2d_1d_0$ of 3 digits.
```
      \let\@tempp\@tempa
      \edef\@tempa{%
```
Here we process d_2 held by `\count5`, that is to say hundreds.
```
        \ifcase\count5 %
        \or cen%
        \or ducen%
        \or trecen%
        \or quadringen%
        \or quingen%
        \or sescen%
        \or septigen%
        \or octingen%
        \or nongen%
        \fi
```
Here we process d_1d_0 held by `\count4` & `\count3`, that is to say tens and units.
```
        \ifnum\count4=0 %
          % x0(0..9)
          \ifnum\count2=3 %
            % Absolute weight zero
            \ifcase\count3 \@tempt
            \or m%
            \or b%
            \or tr%
            \or quadr%
            \or quin\@tempt
            \or sex\@tempt
            \or sep\@tempt
            \or oc\@tempt
            \or non%
            \fi
          \else
```

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

```
              \ifcase\count3 %
              \or \ifnum\count2>\fc@max@weight\else un\fi
              \or d\fc@u@in@duo o%
              \or tre%
              \or quattuor%
              \or quin%
              \or sex%
              \or septen%
              \or octo%
              \or novem%
              \fi
            \fi
          \else
             % x(10..99)
             \ifcase\count3 %
             \or un%
             \or d\fc@u@in@duo o%
             \or tre%
             \or quattuor%
             \or quin%
             \or sex%
             \or septen%
             \or octo%
             \or novem%
             \fi
             \ifcase\count4 %
             \or dec%
             \or vigin\@tempt
             \or trigin\@tempt
             \or quadragin\@tempt
             \or quinquagin\@tempt
             \or sexagin\@tempt
             \or septuagin\@tempt
             \or octogin\@tempt
             \or nonagin\@tempt
             \fi
          \fi
```

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

```
          \@tempm
```

And append previous version of `\@tempa`.

```
          \@tempp
        }%
```

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

```
          \fc@power@of@millia\@tempm{\count2}%
        \fi
        \next
      }%
      \let\@tempa\@empty
      \let\next\@templ
      \@templ
```

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

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

`\fc` @@latin@numeral@pefix Compute a numeral prefix like ‘sémel’, ‘bis’, ‘ter’, ‘quater’, etc. . . I found the algorithm to derive this prefix on Alain Lassine’s site: `http://www.alain.be/Boece/nombres_gargantuesques.html`. First check that the macro is not yet defined.

```
\@ifundefined{fc@@latin@numeral@pefix}{}{%
  \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
    ‘fc@@latin@numeral@pefix’}}
```

Arguments as follows:

#1 input number to be formatted,

#2 outut macro name into which to place the result

```
\def\fc@@latin@numeral@pefix#1#2{%
  {%
    \edef\@tempa{\number#1}%
    \def\fc@unit@weight{0}%
    \expandafter\fc@number@parser\expandafter{\@tempa}%
    \count2=0 %
```

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

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

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

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

Loop body. Three consecutive digits $d_2 d_1 d_0$ are read into counters `\count3`, `\count4`, and `\count5`.

```
        \fc@read@unit{\count3}{\count2}%
        \advance\count2 by 1 %
        \fc@read@unit{\count4}{\count2}%
        \advance\count2 by 1 %
        \fc@read@unit{\count5}{\count2}%
```

```
      \advance\count2 by 1 %
```

Check the use of `duodevicies` instead of `octodecies`.

```
      \let\@tempn\@secondoftwo
      \ifnum\count3>7 %
        \ifnum\count4<\fc@duodeandunde
          \ifnum\count4>0 %
             \let\@tempn\@firstoftwo
           \fi
        \fi
      \fi
      \@tempn
      {% use duodevicies for eighteen
        \advance\count4 by 1 %
        \let\@temps\@secondoftwo
      }{%  do not use duodevicies for eighteen
        \let\@temps\@firstoftwo
      }%
      \let\@tempp\@tempa
      \edef\@tempa{%
        % hundreds
        \ifcase\count5 %
        \expandafter\@gobble
        \or c%
        \or duc%
        \or trec%
        \or quadring%
        \or quing%
        \or sesc%
        \or septing%
        \or octing%
        \or nong%
        \fi
        {enties}%
        \ifnum\count4=0 %
```

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

```
          \ifcase\count3 %
          \or
            \ifnum\count2=3 %
              s\fc@prefix@eaccute emel%
            \else
              \ifnum\count2>\fc@max@weight\else un\fi
            \fi
          \or bis%
          \or ter%
          \or quater%
          \or quinquies%
          \or sexies%
          \or septies%
          \or octies%
          \or novies%
          \fi
        \else
```

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

```
           \ifcase\count3 %
           \or un%
           \or d\fc@u@in@duo o%
           \or ter%
           \or quater%
           \or quin%
           \or sex%
           \or septen%
           \or \@temps{octo}{duod\fc@prefix@eaccute e}% x8 = two before next (x+1)0
           \or \@temps{novem}{und\fc@prefix@eaccute e}% x9 = one before next (x+1)0
           \fi
           \ifcase\count4 %
           % can't get here
           \or d\fc@prefix@eaccute ec%
           \or vic%
           \or tric%
           \or quadrag%
           \or quinquag%
           \or sexag%
           \or septuag%
           \or octog%
           \or nonag%
           \fi
           ies%
        \fi
        % Insert the millies^(n/3) only if one of \count3 \count4 \count5 is non zero
        \@tempm
        % add up previous version of \@tempa
        \@tempp
      }%
```

Concatenate `millies` to `\@tempm` so that it is equal to `millies`$^{n\div3}$ at the next iteration. Here we just have plain concatenation, contrary to cardinal for which a prefix can be used instead.

```
      \let\@tempp\@tempp
      \edef\@tempm{millies\@tempp}%
    \fi
    \next
  }%
  \let\@tempa\@empty
  \let\next\@templ
  \@templ
```

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

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

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.

```
\def\fc@call@opt@arg@second#1#2{%
  \def\@tempb{%
    \ifx[\@tempa
      \def\@tempc[####1]{%
            {#1{####1}{#2}}%
          }%
    \else
      \def\@tempc{{#1{}{#2}}}%
    \fi
    \@tempc
  }%
  \futurelet\@tempa
  \@tempb
}
```

```
\def\fc@call@opt@arg@first#1{%
  \def\@tempb{%
    \ifx[\@tempa
      \def\@tempc[####1]####2{{#1{####1}{####2}}}%
    \else
      \def\@tempc####1{{#1{}{####1}}}%
    \fi
    \@tempc
  }%
  \futurelet\@tempa
  \@tempb
}

\let\fc@call\fc@call@opt@arg@first
```

User API.

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

#1 local options
#2 input number

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

Arguments as follows:

#1 local options
#2 input counter

```
\newcommand*{\@latinnumeralstring}[2]{%
  \setkeys{fcprefix}{#1}%
  \expandafter\let\expandafter
     \@tempa\expandafter\csname c@#2\endcsname
  \expandafter\fc@@latin@numeral@pefix\expandafter{\the\@tempa}\@tempa
  \@tempa
}
```

```
\newcommand*{\latinnumeralstring}{%
  \fc@call\@latinnumeralstring
}
```

```
\newcommand*{\latinnumeralstringnum}{%
  \fc@call\@latinnumeralstringnum
}
```

9.3 fmtcount.sty

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

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{fmtcount}[2013/08/17 v2.03]
\RequirePackage{ifthen}
\RequirePackage{keyval}
\RequirePackage{etoolbox}
\RequirePackage{fcprefix}
```

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

```
\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

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

\padzeroes

```
\padzeroes[⟨n⟩]
```

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

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

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

```
\FCloadlang{⟨language⟩}
```

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

```
\newcount\fc@tmpcatcode
\def\fc@languages{}%
\def\fc@mainlang{}%
\newcommand*{\FCloadlang}[1]{%
  \@FC@iflangloaded{#1}{}%
  {%
    \fc@tmpcatcode=\catcode`\@\relax
    \catcode `\@ 11\relax
    \InputIfFileExists{fc-#1.def}%
    {%
      \ifdefempty{\fc@languages}%
      {%
        \gdef\fc@languages{#1}%
      }%
      {%
         \gappto\fc@languages{,#1}%
      }%
      \gdef\fc@mainlang{#1}%
    }%
    {}%
    \catcode `\@ \fc@tmpcatcode\relax
  }%
}
```

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

`\@FC@iflangloaded{⟨language⟩}{⟨true⟩}{⟨false⟩}`

If fmtcount language definition file `fc-`⟨*language*⟩`.def` has been loaded, do ⟨*true*⟩ otherwise do ⟨*false*⟩

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

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

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

`abelorpolyglossialdf`

`\@fc@loadigbabelldf{⟨language⟩}`

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

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

Load appropriate language definition files:

```
\@fc@loadifbabelorpolyglossialdf{english}
\@fc@loadifbabelorpolyglossialdf{UKenglish}
\@fc@loadifbabelorpolyglossialdf{british}
\@fc@loadifbabelorpolyglossialdf{USenglish}
\@fc@loadifbabelorpolyglossialdf{american}
\@fc@loadifbabelorpolyglossialdf{spanish}
\@fc@loadifbabelorpolyglossialdf{portuges}
\@fc@loadifbabelorpolyglossialdf{french}
\@fc@loadifbabelorpolyglossialdf{frenchb}
\@fc@loadifbabelorpolyglossialdf{francais}
\@fc@loadifbabelorpolyglossialdf{german}%
\@fc@loadifbabelorpolyglossialdf{germanb}%
\@fc@loadifbabelorpolyglossialdf{ngerman}%
\@fc@loadifbabelorpolyglossialdf{ngermanb}%
\@fc@loadifbabelorpolyglossialdf{italian}
```

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

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

`french`

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

```
  \@ifundefined{datefrench}%
  {%
    \PackageError{fmtcount}%
    {Language ‘french’ not defined}%
    {You need to load babel before loading fmtcount}%
  }%
  {%
    \setkeys{fcfrench}{#1}%
  }%
}
```

fmtord Key to determine how to display the ordinal

```
\define@key{fmtcount}{fmtord}{%
  \ifthenelse{\equal{#1}{level}
          \or\equal{#1}{raise}
          \or\equal{#1}{user}}%
  {%
    \def\fmtcount@fmtord{#1}%
  }%
  {%
    \PackageError{fmtcount}%
    {Invalid value ‘#1’ to fmtord key}%
    {Option ‘fmtord’ can only take the values ‘level’, ‘raise’
     or ‘user’}%
  }%
}
```

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

```
\newif\iffmtord@abbrv
\fmtord@abbrvfalse
\define@key{fmtcount}{abbrv}[true]{%
  \ifthenelse{\equal{#1}{true}\or\equal{#1}{false}}%
  {%
    \csname fmtord@abbrv#1\endcsname
  }%
  {%
    \PackageError{fmtcount}%
    {Invalid value ‘#1’ to fmtord key}%
    {Option ‘fmtord’ can only take the values ‘true’ or
     ‘false’}%
  }%
}
```

prefix

```
\define@key{fmtcount}{prefix}[scale=long]{%
  \RequirePackage{fmtprefix}%
  \fmtprefixsetoption{#1}%
}
```

\fmtcountsetoptions Define command to set options.

```
\newcommand*{\fmtcountsetoptions}[1]{%
  \def\fmtcount@fmtord{}%
  \setkeys{fmtcount}{#1}%
  \@ifundefined{datefrench}{}%
  {%
    \edef\@ordinalstringMfrench{\noexpand
      \csname @ordinalstringMfrench\fmtcount@french\noexpand\endcsname}%
    \edef\@ordinalstringFfrench{\noexpand
      \csname @ordinalstringFfrench\fmtcount@french\noexpand\endcsname}%
    \edef\@OrdinalstringMfrench{\noexpand
      \csname @OrdinalstringMfrench\fmtcount@french\noexpand\endcsname}%
    \edef\@OrdinalstringFfrench{\noexpand
      \csname @OrdinalstringFfrench\fmtcount@french\noexpand\endcsname}%
    \edef\@numberstringMfrench{\noexpand
      \csname @numberstringMfrench\fmtcount@french\noexpand\endcsname}%
    \edef\@numberstringFfrench{\noexpand
      \csname @numberstringFfrench\fmtcount@french\noexpand\endcsname}%
    \edef\@NumberstringMfrench{\noexpand
      \csname @NumberstringMfrench\fmtcount@french\noexpand\endcsname}%
    \edef\@NumberstringFfrench{\noexpand
      \csname @NumberstringFfrench\fmtcount@french\noexpand\endcsname}%
  }%
  \ifthenelse{\equal{\fmtcount@fmtord}{level}}%
  {%
    \renewcommand{\fmtord}[1]{##1}%
  }%
  {%
    \ifthenelse{\equal{\fmtcount@fmtord}{raise}}%
    {%
      \renewcommand{\fmtord}[1]{\textsuperscript{##1}}%
    }%
    {%
    }%
  }
}
```

Load confguration 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.

```
\InputIfFileExists{fmtcount.cfg}%
{%
  \PackageInfo{fmtcount}{Using configuration file fmtcount.cfg}%
}%
{%
}
```

level

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

raise

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

Process package options

```
\ProcessOptions
```

\@modulo

\@modulo{⟨*count reg*⟩}{⟨*n*⟩}

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

```
\newcount\@DT@modctr
\def\@modulo#1#2{%
  \@DT@modctr=#1\relax
  \divide \@DT@modctr by #2\relax
  \multiply \@DT@modctr by #2\relax
  \advance #1 by -\@DT@modctr
}
```

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

```
\newcount\@ordinalctr
\newcount\@orgargctr
\newcount\@strctr
\newcount\@tmpstrctr
```

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

```
\newif\if@DT@padzeroes
\newcount\@DT@loopN
\newcount\@DT@X
```

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

```
\newcommand*{\@binary}[1]{%
  \@DT@padzeroestrue
  \@DT@loopN=17\relax
  \@strctr=\@DT@loopN
  \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by 1}%
  \@strctr=65536\relax
  \@DT@X=#1\relax
  \loop
    \@DT@modctr=\@DT@X
    \divide\@DT@modctr by \@strctr
    \ifthenelse{\boolean{@DT@padzeroes}
       \and \(\@DT@modctr=0\)
       \and \(\@DT@loopN>\c@padzeroesN\)}%
    {}%
    {\the\@DT@modctr}%
    \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
    \multiply\@DT@modctr by \@strctr
    \advance\@DT@X by -\@DT@modctr
    \divide\@strctr by 2\relax
    \advance\@DT@loopN by -1\relax
  \ifnum\@strctr>1
  \repeat
  \the\@DT@X
}

\let\binarynum=\@binary
```

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

```
\newcommand*{\@octal}[1]{%
  \ifnum#1>32768
    \PackageError{fmtcount}%
    {Value of counter too large for \protect\@octal}
    {Maximum value 32768}
  \else
  \@DT@padzeroestrue
  \@DT@loopN=6\relax
  \@strctr=\@DT@loopN
  \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by 1}%
  \@strctr=32768\relax
  \@DT@X=#1\relax
  \loop
    \@DT@modctr=\@DT@X
    \divide\@DT@modctr by \@strctr
    \ifthenelse{\boolean{@DT@padzeroes}
       \and \(\@DT@modctr=0\)
       \and \(\@DT@loopN>\c@padzeroesN\)}%
    {}{\the\@DT@modctr}%
    \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
    \multiply\@DT@modctr by \@strctr
    \advance\@DT@X by -\@DT@modctr
    \divide\@strctr by 8\relax
    \advance\@DT@loopN by -1\relax
  \ifnum\@strctr>1
  \repeat
  \the\@DT@X
  \fi
}
\let\octalnum=\@octal
```

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

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

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

```
\newcommand*{\@hexadecimal}[1]{%
  \@DT@padzeroestrue
  \@DT@loopN=5\relax
  \@strctr=\@DT@loopN
  \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by 1}%
  \@strctr=65536\relax
  \@DT@X=#1\relax
  \loop
    \@DT@modctr=\@DT@X
    \divide\@DT@modctr by \@strctr
    \ifthenelse{\boolean{@DT@padzeroes}
      \and \(\@DT@modctr=0\)
      \and \(\@DT@loopN>\c@padzeroesN\)}
    {}{\@@hexadecimal\@DT@modctr}%
    \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
    \multiply\@DT@modctr by \@strctr
    \advance\@DT@X by -\@DT@modctr
    \divide\@strctr by 16\relax
    \advance\@DT@loopN by -1\relax
  \ifnum\@strctr>1
  \repeat
  \@@hexadecimal\@DT@X
}
\let\hexadecimalnum=\@hexadecimal
```

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

```
\newcommand*{\@@Hexadecimal}[1]{%
  \ifcase#10\or1\or2\or3\or4\or5\or6\or
  7\or8\or9\or A\or B\or C\or D\or E\or F\fi
}
```

\Hexadecimalnum Uppercase hexadecimal

```
\newcommand*{\@Hexadecimal}[1]{%
  \@DT@padzeroestrue
  \@DT@loopN=5\relax
  \@strctr=\@DT@loopN
  \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by 1}%
  \@strctr=65536\relax
  \@DT@X=#1\relax
  \loop
    \@DT@modctr=\@DT@X
    \divide\@DT@modctr by \@strctr
    \ifthenelse{\boolean{@DT@padzeroes}
      \and \(\@DT@modctr=0\)
      \and \(\@DT@loopN>\c@padzeroesN\)}%
    {}{\@@Hexadecimal\@DT@modctr}%
    \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
    \multiply\@DT@modctr by \@strctr
    \advance\@DT@X by -\@DT@modctr
    \divide\@strctr by 16\relax
    \advance\@DT@loopN by -1\relax
  \ifnum\@strctr>1
  \repeat
  \@@Hexadecimal\@DT@X
}

\let\Hexadecimalnum=\@Hexadecimal
```

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

```
\newcommand*{\@aaalph}[1]{%
  \@DT@loopN=#1\relax
  \advance\@DT@loopN by -1\relax
  \divide\@DT@loopN by 26\relax
  \@DT@modctr=\@DT@loopN
  \multiply\@DT@modctr by 26\relax
  \@DT@X=#1\relax
  \advance\@DT@X by -1\relax
  \advance\@DT@X by -\@DT@modctr
  \advance\@DT@loopN by 1\relax
  \advance\@DT@X by 1\relax
  \loop
    \@alph\@DT@X
    \advance\@DT@loopN by -1\relax
  \ifnum\@DT@loopN>0
  \repeat
}

\let\aaalphnum=\@aaalph
```

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

```
\newcommand*{\@AAAlph}[1]{%
  \@DT@loopN=#1\relax
  \advance\@DT@loopN by -1\relax
  \divide\@DT@loopN by 26\relax
  \@DT@modctr=\@DT@loopN
  \multiply\@DT@modctr by 26\relax
  \@DT@X=#1\relax
  \advance\@DT@X by -1\relax
  \advance\@DT@X by -\@DT@modctr
  \advance\@DT@loopN by 1\relax
  \advance\@DT@X by 1\relax
  \loop
    \@Alph\@DT@X
    \advance\@DT@loopN by -1\relax
  \ifnum\@DT@loopN>0
  \repeat
}

\let\AAAlphnum=\@AAAlph
```

`\abalphnum` Lowercase alphabetical representation

```
\newcommand*{\@abalph}[1]{%
  \ifnum#1>17576\relax
    \PackageError{fmtcount}%
    {Value of counter too large for \protect\@abalph}%
    {Maximum value 17576}%
  \else
    \@DT@padzeroestrue
    \@strctr=17576\relax
    \@DT@X=#1\relax
    \advance\@DT@X by -1\relax
    \loop
      \@DT@modctr=\@DT@X
      \divide\@DT@modctr by \@strctr
      \ifthenelse{\boolean{@DT@padzeroes}
        \and \(\@DT@modctr=1\)}%
      {}{\@alph\@DT@modctr}%
      \ifnum\@DT@modctr=1\else\@DT@padzeroesfalse\fi
      \multiply\@DT@modctr by \@strctr
      \advance\@DT@X by -\@DT@modctr
      \divide\@strctr by 26\relax
    \ifnum\@strctr>1
    \repeat
    \advance\@DT@X by 1\relax
    \@alph\@DT@X
  \fi
}

\let\abalphnum=\@abalph
```

`\ABAlphnum` Uppercase alphabetical representation

```
\newcommand*{\@ABAlph}[1]{%
  \ifnum#1>17576\relax
    \PackageError{fmtcount}%
   {Value of counter too large for \protect\@ABAlph}%
   {Maximum value 17576}%
  \else
    \@DT@padzeroestrue
    \@strctr=17576\relax
    \@DT@X=#1\relax
    \advance\@DT@X by -1\relax
    \loop
      \@DT@modctr=\@DT@X
      \divide\@DT@modctr by \@strctr
      \ifthenelse{\boolean{@DT@padzeroes}\and
      \(\@DT@modctr=1\)}{}{\@Alph\@DT@modctr}%
      \ifnum\@DT@modctr=1\else\@DT@padzeroesfalse\fi
      \multiply\@DT@modctr by \@strctr
      \advance\@DT@X by -\@DT@modctr
      \divide\@strctr by 26\relax
    \ifnum\@strctr>1
    \repeat
    \advance\@DT@X by 1\relax
    \@Alph\@DT@X
  \fi
}

\let\ABAlphnum=\@ABAlph
```

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

```
\def\@fmtc@count#1#2\relax{%
  \if\relax#1%
  \else
    \advance\@strctr by 1\relax
    \@fmtc@count#2\relax
  \fi
}
```

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

```
\newcommand{\@decimal}[1]{%
  \@strctr=0\relax
  \expandafter\@fmtc@count\number#1\relax
  \@DT@loopN=\c@padzeroesN
  \advance\@DT@loopN by -\@strctr
  \ifnum\@DT@loopN>0\relax
    \@strctr=0\relax
    \whiledo{\@strctr < \@DT@loopN}{0\advance\@strctr by 1\relax}%
  \fi
  \number#1\relax
}

\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.

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

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

```
\@ifundefined{ordinal}
 {\let\ordinal\FCordinal}%
 {%
   \PackageWarning{fmtcount}%
  {\string\ordinal \space already defined use
   \string\FCordinal \space instead.}
 }
```

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

```
\newcommand*{\ordinalnum}[1]{%
  \new@ifnextchar[%
  {\@ordinalnum{#1}}%
  {\@ordinalnum{#1}[m]}%
}
```

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

```
\def\@ordinalnum#1[#2]{%
  {%
    \ifthenelse{\equal{#2}{f}}%
    {%
      \protect\@ordinalF{#1}{\@fc@ordstr}%
    }%
    {%
      \ifthenelse{\equal{#2}{n}}%
      {%
        \protect\@ordinalN{#1}{\@fc@ordstr}%
      }%
      {%
        \ifthenelse{\equal{#2}{m}}%
        {}%
        {%
```

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

```
          \PackageError{fmtcount}%
           {Invalid gender option '#2'}%
           {Available options are m, f or n}%
        }%
        \protect\@ordinalM{#1}{\@fc@ordstr}%
      }%
    }%
    \@fc@ordstr
  }%
}
```

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

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

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

```
\newcommand*{\storeordinalnum}[2]{%
  \@ifnextchar[%
  {\@storeordinalnum{#1}{#2}}%
  {\@storeordinalnum{#1}{#2}[m]}%
}
```

\@storeordinalnum Store ordinal according to gender:

```
\def\@storeordinalnum#1#2[#3]{%
  \ifthenelse{\equal{#3}{f}}%
  {%
    \protect\@ordinalF{#2}{\@fc@ord}
  }%
  {%
    \ifthenelse{\equal{#3}{n}}%
    {%
      \protect\@ordinalN{#2}{\@fc@ord}%
    }%
    {%
      \ifthenelse{\equal{#3}{m}}%
      {}%
      {%
        \PackageError{fmtcount}%
        {Invalid gender option '#3'}%
        {Available options are m or f}%
      }%
      \protect\@ordinalM{#2}{\@fc@ord}%
    }%
  }%
  \expandafter\let\csname @fcs@#1\endcsname\@fc@ord
}
```

\FMCuse Get stored information:

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

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

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

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

```
\newcommand{\ordinalstringnum}[1]{%
  \new@ifnextchar[%
  {\@ordinal@string{#1}}%
  {\@ordinal@string{#1}[m]}%
}
```

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

```
\def\@ordinal@string#1[#2]{%
  {%
    \ifthenelse{\equal{#2}{f}}%
    {%
      \protect\@ordinalstringF{#1}{\@fc@ordstr}%
    }%
    {%
      \ifthenelse{\equal{#2}{n}}%
      {%
        \protect\@ordinalstringN{#1}{\@fc@ordstr}%
      }%
      {%
        \ifthenelse{\equal{#2}{m}}%
        {}%
        {%
          \PackageError{fmtcount}%
          {Invalid gender option ‘#2’ to \string\ordinalstring}%
          {Available options are m, f or f}%
        }%
        \protect\@ordinalstringM{#1}{\@fc@ordstr}%
      }%
    }%
    \@fc@ordstr
  }%
}
```

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

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

```
  \expandafter\protect\expandafter\storeordinalstringnum{#1}{%
    \expandafter\the\csname c@#2\endcsname}%
}
```

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

```
\newcommand*{\storeordinalstringnum}[2]{%
  \@ifnextchar[%
  {\@store@ordinal@string{#1}{#2}}%
  {\@store@ordinal@string{#1}{#2}[m]}%
}
```

tore@ordinal@string Store textual representation of number according to gender.

```
\def\@store@ordinal@string#1#2[#3]{%
  \ifthenelse{\equal{#3}{f}}%
  {%
    \protect\@ordinalstringF{#2}{\@fc@ordstr}%
  }%
  {%
    \ifthenelse{\equal{#3}{n}}%
    {%
      \protect\@ordinalstringN{#2}{\@fc@ordstr}%
    }%
    {%
      \ifthenelse{\equal{#3}{m}}%
      {}%
      {%
        \PackageError{fmtcount}%
        {Invalid gender option ‘#3’ to \string\ordinalstring}%
        {Available options are m, f or n}%
      }%
      \protect\@ordinalstringM{#2}{\@fc@ordstr}%
    }%
  }%
  \expandafter\let\csname @fcs@#1\endcsname\@fc@ordstr
}
```

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

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

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

```
\newcommand*{\Ordinalstringnum}[1]{%
  \new@ifnextchar[%
  {\@Ordinal@string{#1}}%
  {\@Ordinal@string{#1}[m]}%
}
```

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

```
\def\@Ordinal@string#1[#2]{%
  {%
    \ifthenelse{\equal{#2}{f}}%
    {%
      \protect\@OrdinalstringF{#1}{\@fc@ordstr}%
    }%
    {%
      \ifthenelse{\equal{#2}{n}}%
      {%
        \protect\@OrdinalstringN{#1}{\@fc@ordstr}%
      }%
      {%
        \ifthenelse{\equal{#2}{m}}%
        {}%
        {%
          \PackageError{fmtcount}%
          {Invalid gender option ‘#2’}%
          {Available options are m, f or n}%
        }%
        \protect\@OrdinalstringM{#1}{\@fc@ordstr}%
      }%
    }%
    \@fc@ordstr
  }%
}
```

\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.

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

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.

```
\newcommand*{\storeOrdinalstringnum}[2]{%
  \@ifnextchar[%
  {\@store@Ordinal@string{#1}{#2}}%
  {\@store@Ordinal@string{#1}{#2}[m]}%
}
```

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

```
\def\@store@Ordinal@string#1#2[#3]{%
  \ifthenelse{\equal{#3}{f}}%
  {%
    \protect\@OrdinalstringF{#2}{\@fc@ordstr}%
  }%
  {%
    \ifthenelse{\equal{#3}{n}}%
    {%
      \protect\@OrdinalstringN{#2}{\@fc@ordstr}%
    }%
    {%
      \ifthenelse{\equal{#3}{m}}%
      {}%
      {%
        \PackageError{fmtcount}%
        {Invalid gender option ‘#3’}%
        {Available options are m or f}%
      }%
      \protect\@OrdinalstringM{#2}{\@fc@ordstr}%
    }%
  }%
  \expandafter\let\csname @fcs@#1\endcsname\@fc@ordstr
}
```

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

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

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

```
\newcommand*{\storeORDINALstringnum}[2]{%
  \@ifnextchar[%
  {\@store@ORDINAL@string{#1}{#2}}%
  {\@store@ORDINAL@string{#1}{#2}[m]}%
}
```

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

```
\def\@store@ORDINAL@string#1#2[#3]{%
  \ifthenelse{\equal{#3}{f}}%
  {%
    \protect\@ordinalstringF{#2}{\@fc@ordstr}%
  }%
  {%
    \ifthenelse{\equal{#3}{n}}%
    {%
      \protect\@ordinalstringN{#2}{\@fc@ordstr}%
    }%
    {%
      \ifthenelse{\equal{#3}{m}}%
      {}%
      {%
        \PackageError{fmtcount}%
        {Invalid gender option ‘#3’}%
        {Available options are m or f}%
      }%
      \protect\@ordinalstringM{#2}{\@fc@ordstr}%
    }%
  }%
  \expandafter\edef\csname @fcs@#1\endcsname{%
    \noexpand\MakeUppercase{\@fc@ordstr}%
  }%
}
```

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

```
\newcommand*{\ORDINALstring}[1]{%
  \expandafter\protect\expandafter\ORDINALstringnum{%
    \expandafter\the\csname c@#1\endcsname
  }%
}
```

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

```
\newcommand*{\ORDINALstringnum}[1]{%
  \new@ifnextchar[%
  {\@ORDINAL@string{#1}}%
  {\@ORDINAL@string{#1}[m]}%
}
```

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

```
\def\@ORDINAL@string#1[#2]{%
  {%
    \ifthenelse{\equal{#2}{f}}%
    {%
      \protect\@ordinalstringF{#1}{\@fc@ordstr}%
    }%
    {%
      \ifthenelse{\equal{#2}{n}}%
      {%
        \protect\@ordinalstringN{#1}{\@fc@ordstr}%
      }%
      {%
        \ifthenelse{\equal{#2}{m}}%
        {}%
        {%
          \PackageError{fmtcount}%
          {Invalid gender option ‘#2’}%
          {Available options are m, f or n}%
        }%
        \protect\@ordinalstringM{#1}{\@fc@ordstr}%
      }%
    }%
    \MakeUppercase{\@fc@ordstr}%
  }%
}
```

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

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

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

```
\newcommand{\storenumberstringnum}[2]{%
  \@ifnextchar[%
  {\@store@number@string{#1}{#2}}%
  {\@store@number@string{#1}{#2}[m]}%
}
```

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

```
\def\@store@number@string#1#2[#3]{%
  \ifthenelse{\equal{#3}{f}}%
  {%
    \protect\@numberstringF{#2}{\@fc@numstr}%
  }%
  {%
    \ifthenelse{\equal{#3}{n}}%
    {%
      \protect\@numberstringN{#2}{\@fc@numstr}%
    }%
    {%
      \ifthenelse{\equal{#3}{m}}%
      {}%
      {%
        \PackageError{fmtcount}
        {Invalid gender option '#3'}%
        {Available options are m, f or n}%
      }%
      \protect\@numberstringM{#2}{\@fc@numstr}%
    }%
  }%
  \expandafter\let\csname @fcs@#1\endcsname\@fc@numstr
}
```

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

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

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

```
\newcommand*{\numberstringnum}[1]{%
  \new@ifnextchar[%
  {\@number@string{#1}}%
  {\@number@string{#1}[m]}%
}
```

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

```
\def\@number@string#1[#2]{%
  {%
    \ifthenelse{\equal{#2}{f}}%
    {%
      \protect\@numberstringF{#1}{\@fc@numstr}%
    }%
    {%
      \ifthenelse{\equal{#2}{n}}%
      {%
        \protect\@numberstringN{#1}{\@fc@numstr}%
      }%
      {%
        \ifthenelse{\equal{#2}{m}}%
        {}%
        {%
          \PackageError{fmtcount}%
          {Invalid gender option '#2'}%
          {Available options are m, f or n}%
        }%
        \protect\@numberstringM{#1}{\@fc@numstr}%
      }%
    }%
    \@fc@numstr
  }%
}
```

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

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

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

```
\newcommand{\storeNumberstringnum}[2]{%
  \@ifnextchar[%
  {\@store@Number@string{#1}{#2}}%
  {\@store@Number@string{#1}{#2}[m]}%
}
```

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

```
\def\@store@Number@string#1#2[#3]{%
  \ifthenelse{\equal{#3}{f}}%
  {%
    \protect\@NumberstringF{#2}{\@fc@numstr}%
  }%
  {%
    \ifthenelse{\equal{#3}{n}}%
    {%
      \protect\@NumberstringN{#2}{\@fc@numstr}%
    }%
    {%
      \ifthenelse{\equal{#3}{m}}%
      {}%
      {%
        \PackageError{fmtcount}%
        {Invalid gender option '#3'}%
        {Available options are m, f or n}%
      }%
      \protect\@NumberstringM{#2}{\@fc@numstr}%
    }%
  }%
  \expandafter\let\csname @fcs@#1\endcsname\@fc@numstr
}
```

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

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

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

```
\newcommand*{\Numberstringnum}[1]{%
  \new@ifnextchar[%
  {\@Number@string{#1}}%
  {\@Number@string{#1}[m]}%
}
```

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

```
\def\@Number@string#1[#2]{%
  {%
    \ifthenelse{\equal{#2}{f}}%
    {%
      \protect\@NumberstringF{#1}{\@fc@numstr}%
    }%
    {%
      \ifthenelse{\equal{#2}{n}}%
      {%
        \protect\@NumberstringN{#1}{\@fc@numstr}%
      }%
      {%
        \ifthenelse{\equal{#2}{m}}%
        {}%
        {%
          \PackageError{fmtcount}%
          {Invalid gender option ‘#2’}%
          {Available options are m, f or n}%
        }%
        \protect\@NumberstringM{#1}{\@fc@numstr}%
      }%
    }%
    \@fc@numstr
  }%
}
```

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

```
\newcommand{\storeNUMBERstring}[2]{%
  \expandafter\protect\expandafter\storeNUMBERstringnum{#1}{%
    \expandafter\the\csname c@#2\endcsname}%
}
```

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

```
\newcommand{\storeNUMBERstringnum}[2]{%
  \@ifnextchar[%
  {\@store@NUMBER@string{#1}{#2}}%
  {\@store@NUMBER@string{#1}{#2}[m]}%
}
```

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

```
\def\@store@NUMBER@string#1#2[#3]{%
  \ifthenelse{\equal{#3}{f}}%
  {%
    \protect\@numberstringF{#2}{\@fc@numstr}%
  }%
  {%
    \ifthenelse{\equal{#3}{n}}%
    {%
      \protect\@numberstringN{#2}{\@fc@numstr}%
    }%
    {%
      \ifthenelse{\equal{#3}{m}}%
      {}%
      {%
```

```
        \PackageError{fmtcount}%
        {Invalid gender option ‘#3’}%
        {Available options are m or f}%
      }%
      \protect\@numberstringM{#2}{\@fc@numstr}%
    }%
  }%
  \expandafter\edef\csname @fcs@#1\endcsname{%
    \noexpand\MakeUppercase{\@fc@numstr}%
  }%
}
```

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

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

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

```
\newcommand*{\NUMBERstringnum}[1]{%
  \new@ifnextchar[%
  {\@NUMBER@string{#1}}%
  {\@NUMBER@string{#1}[m]}%
}
```

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

```
\def\@NUMBER@string#1[#2]{%
  {%
    \ifthenelse{\equal{#2}{f}}%
    {%
      \protect\@numberstringF{#1}{\@fc@numstr}%
    }%
    {%
      \ifthenelse{\equal{#2}{n}}%
      {%
         \protect\@numberstringN{#1}{\@fc@numstr}%
      }%
      {%
        \ifthenelse{\equal{#2}{m}}%
        {}%
        {%
          \PackageError{fmtcount}%
          {Invalid gender option ‘#2’}%
          {Available options are m, f or n}%
        }%
        \protect\@numberstringM{#1}{\@fc@numstr}%
      }%
    }%
    \MakeUppercase{\@fc@numstr}%
  }%
}
```

\binary Number representations in other bases. Binary:

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

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

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

\AAAlph As before, but upper case.

```
\providecommand*{\AAAlph}[1]{%
  \expandafter\protect\expandafter\@AAAlph{%
    \expandafter\the\csname c@#1\endcsname}%
}
```

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

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

\ABAlph As above, but upper case.

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

\hexadecimal Hexadecimal:

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

\Hexadecimal As above, but in upper case.

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

\octal Octal:

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

```
  \expandafter\protect\expandafter\@octal{%
    \expandafter\the\csname c@#1\endcsname}%
}
```

\decimal Decimal:

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

9.4 Multilinguage 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.

```
\def\@setdef@ultfmtcount{%
  \@ifundefined{@ordinalMenglish}{\FCloadlang{english}}{}%
  \def\@ordinalstringM{\@ordinalstringMenglish}%
  \let\@ordinalstringF=\@ordinalstringMenglish
  \let\@ordinalstringN=\@ordinalstringMenglish
  \def\@OrdinalstringM{\@OrdinalstringMenglish}%
  \let\@OrdinalstringF=\@OrdinalstringMenglish
  \let\@OrdinalstringN=\@OrdinalstringMenglish
  \def\@numberstringM{\@numberstringMenglish}%
  \let\@numberstringF=\@numberstringMenglish
  \let\@numberstringN=\@numberstringMenglish
  \def\@NumberstringM{\@NumberstringMenglish}%
  \let\@NumberstringF=\@NumberstringMenglish
  \let\@NumberstringN=\@NumberstringMenglish
  \def\@ordinalM{\@ordinalMenglish}%
  \let\@ordinalF=\@ordinalM
  \let\@ordinalN=\@ordinalM
}
```

\fc@multiling changes2.022012-10-24new `\fc@multiling{`⟨*name*⟩`}{`⟨*gender*⟩`}`

```
\newcommand*{\fc@multiling}[2]{%
  \ifcsundef{@#1#2\languagename}%
  {% try loading it
     \FCloadlang{\languagename}%
  }%
  {%
  }%
  \ifcsundef{@#1#2\languagename}%
  {%
    \PackageWarning{fmtcount}%
    {No support for \expandafter\string\csname#1\endcsname\space for
     language '\languagename'}%
    \ifthenelse{\equal{\languagename}{\fc@mainlang}}%
    {%
      \FCloadlang{english}%
    }%
    {%
    }%
    \ifcsdef{@#1#2\fc@mainlang}%
    {%
      \csuse{@#1#2\fc@mainlang}%
    }%
    {%
      \PackageWarningNoLine{fmtcount}%
      {No languages loaded at all! Loading english definitions}%
      \FCloadlang{english}%
      \def\fc@mainlang{english}%
      \csuse{@#1#2english}%
    }%
  }%
  {%
    \csuse{@#1#2\languagename}%
  }%
}
```

@mulitling@fmtcount

This defines the number and ordinal string macros to use \languagename:

```
\def\@set@mulitling@fmtcount{%
```

The masculine version of \numberstring:

```
  \def\@numberstringM{%
    \fc@multiling{numberstring}{M}%
  }%
```

The feminine version of \numberstring:

```
  \def\@numberstringF{%
    \fc@multiling{numberstring}{F}%
  }%
```

The neuter version of \numberstring:

```
  \def\@numberstringN{%
    \fc@multiling{numberstring}{N}%
  }%
```

The masculine version of \Numberstring:

```
  \def\@NumberstringM{%
    \fc@multiling{Numberstring}{M}%
  }%
```

The feminine version of \Numberstring:

```
  \def\@NumberstringF{%
    \fc@multiling{Numberstring}{F}%
  }%
```

The neuter version of \Numberstring:

```
  \def\@NumberstringN{%
```

```
    \fc@multiling{Numberstring}{N}%
  }%
```

The masculine version of `\ordinal`:

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

The feminine version of `\ordinal`:

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

The neuter version of `\ordinal`:

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

The masculine version of `\ordinalstring`:

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

The feminine version of `\ordinalstring`:

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

The neuter version of `\ordinalstring`:

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

The masculine version of `\Ordinalstring`:

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

The feminine version of `\Ordinalstring`:

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

The neuter version of `\Ordinalstring`:

```
  \def\@OrdinalstringN{%
    \fc@multiling{Ordinalstring}{N}%
  }%
}
```

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

```
\@ifpackageloaded{babel}%
{%
  \@set@mulitling@fmtcount
}%
{%
  \@ifpackageloaded{ngerman}%
  {%
    \FCloadlang{ngerman}%
    \@set@mulitling@fmtcount
  }%
  {%
    \@setdef@ultfmtcount
  }%
}
```

Backwards compatibility:

```
\let\@ordinal=\@ordinalM
\let\@ordinalstring=\@ordinalstringM
\let\@Ordinalstring=\@OrdinalstringM
\let\@numberstring=\@numberstringM
\let\@Numberstring=\@NumberstringM
```

9.4.1 fc-american.def

American English definitions

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

Loaded fc-USenglish.def if not already loaded

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

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

```
\global\let\@ordinalMamerican\@ordinalMUSenglish
\global\let\@ordinalFamerican\@ordinalMUSenglish
\global\let\@ordinalNamerican\@ordinalMUSenglish
\global\let\@numberstringMamerican\@numberstringMUSenglish
\global\let\@numberstringFamerican\@numberstringMUSenglish
\global\let\@numberstringNamerican\@numberstringMUSenglish
\global\let\@NumberstringMamerican\@NumberstringMUSenglish
\global\let\@NumberstringFamerican\@NumberstringMUSenglish
\global\let\@NumberstringNamerican\@NumberstringMUSenglish
\global\let\@ordinalstringMamerican\@ordinalstringMUSenglish
\global\let\@ordinalstringFamerican\@ordinalstringMUSenglish
\global\let\@ordinalstringNamerican\@ordinalstringMUSenglish
\global\let\@OrdinalstringMamerican\@OrdinalstringMUSenglish
\global\let\@OrdinalstringFamerican\@OrdinalstringMUSenglish
\global\let\@OrdinalstringNamerican\@OrdinalstringMUSenglish
```

9.4.2 fc-british.def

British definitions

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

Load fc-english.def, if not already loaded

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

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

```
\global\let\@ordinalMbritish\@ordinalMenglish
\global\let\@ordinalFbritish\@ordinalMenglish
\global\let\@ordinalNbritish\@ordinalMenglish
\global\let\@numberstringMbritish\@numberstringMenglish
\global\let\@numberstringFbritish\@numberstringMenglish
\global\let\@numberstringNbritish\@numberstringMenglish
\global\let\@NumberstringMbritish\@NumberstringMenglish
\global\let\@NumberstringFbritish\@NumberstringMenglish
\global\let\@NumberstringNbritish\@NumberstringMenglish
\global\let\@ordinalstringMbritish\@ordinalstringMenglish
\global\let\@ordinalstringFbritish\@ordinalstringMenglish
\global\let\@ordinalstringNbritish\@ordinalstringMenglish
\global\let\@OrdinalstringMbritish\@OrdinalstringMenglish
\global\let\@OrdinalstringFbritish\@OrdinalstringMenglish
\global\let\@OrdinalstringNbritish\@OrdinalstringMenglish
```

9.4.3 fc-english.def

English definitions

```
\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.

```
\gdef\@ordinalMenglish#1#2{%
\def\@fc@ord{}%
\@orgargctr=#1\relax
\@ordinalctr=#1%
\@modulo{\@ordinalctr}{100}%
\ifnum\@ordinalctr=11\relax
  \def\@fc@ord{th}%
\else
  \ifnum\@ordinalctr=12\relax
    \def\@fc@ord{th}%
  \else
    \ifnum\@ordinalctr=13\relax
      \def\@fc@ord{th}%
    \else
      \@modulo{\@ordinalctr}{10}%
      \ifcase\@ordinalctr
        \def\@fc@ord{th}%      case 0
        \or \def\@fc@ord{st}%  case 1
        \or \def\@fc@ord{nd}%  case 2
        \or \def\@fc@ord{rd}%  case 3
      \else
        \def\@fc@ord{th}%      default case
      \fi
    \fi
  \fi
\fi
\edef#2{\number#1\relax\noexpand\fmtord{\@fc@ord}}%
}%
```

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

```
\global\let\@ordinalFenglish=\@ordinalMenglish
\global\let\@ordinalNenglish=\@ordinalMenglish
```

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

```
\gdef\@@unitstringenglish#1{%
  \ifcase#1\relax
    zero%
    \or one%
    \or two%
    \or three%
    \or four%
    \or five%
    \or six%
    \or seven%
    \or eight%
    \or nine%
\fi
}%
```

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

```
\gdef\@@tenstringenglish#1{%
  \ifcase#1\relax
    \or ten%
    \or twenty%
    \or thirty%
    \or forty%
    \or fifty%
    \or sixty%
    \or seventy%
    \or eighty%
    \or ninety%
  \fi
}%
```

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

```
\gdef\@@teenstringenglish#1{%
  \ifcase#1\relax
    ten%
    \or eleven%
    \or twelve%
    \or thirteen%
    \or fourteen%
    \or fifteen%
    \or sixteen%
    \or seventeen%
    \or eighteen%
    \or nineteen%
  \fi
}%
```

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

```
\gdef\@@Unitstringenglish#1{%
  \ifcase#1\relax
    Zero%
    \or One%
    \or Two%
    \or Three%
    \or Four%
    \or Five%
    \or Six%
    \or Seven%
    \or Eight%
    \or Nine%
  \fi
}%
```

The tens:

```
\gdef\@@Tenstringenglish#1{%
  \ifcase#1\relax
    \or Ten%
    \or Twenty%
    \or Thirty%
    \or Forty%
    \or Fifty%
    \or Sixty%
    \or Seventy%
    \or Eighty%
    \or Ninety%
  \fi
}%
```

The teens:

```
\gdef\@@Teenstringenglish#1{%
  \ifcase#1\relax
    Ten%
    \or Eleven%
    \or Twelve%
    \or Thirteen%
    \or Fourteen%
    \or Fifteen%
    \or Sixteen%
    \or Seventeen%
    \or Eighteen%
    \or Nineteen%
  \fi
}%
```

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

```
\gdef\@@numberstringenglish#1#2{%
\ifnum#1>99999
\PackageError{fmtcount}{Out of range}%
{This macro only works for values less than 100000}%
\else
\ifnum#1<0
\PackageError{fmtcount}{Negative numbers not permitted}%
{This macro does not work for negative numbers, however
you can try typing "minus" first, and then pass the modulus of
this number}%
\fi
\fi
\def#2{}%
\@strctr=#1\relax \divide\@strctr by 1000\relax
\ifnum\@strctr>9
  \divide\@strctr by 10
  \ifnum\@strctr>1\relax
    \let\@@fc@numstr#2\relax
    \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
    \@strctr=#1 \divide\@strctr by 1000\relax
    \@modulo{\@strctr}{10}%
    \ifnum\@strctr>0\relax
      \let\@@fc@numstr#2\relax
      \edef#2{\@@fc@numstr-\@unitstring{\@strctr}}%
    \fi
  \else
    \@strctr=#1\relax
    \divide\@strctr by 1000\relax
    \@modulo{\@strctr}{10}%
    \let\@@fc@numstr#2\relax
    \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
  \fi
  \let\@@fc@numstr#2\relax
  \edef#2{\@@fc@numstr\ \@thousand}%
\else
  \ifnum\@strctr>0\relax
    \let\@@fc@numstr#2\relax
    \edef#2{\@@fc@numstr\@unitstring{\@strctr}\ \@thousand}%
  \fi
\fi
\@strctr=#1\relax \@modulo{\@strctr}{1000}%
\divide\@strctr by 100
\ifnum\@strctr>0\relax
   \ifnum#1>1000\relax
      \let\@@fc@numstr#2\relax
      \edef#2{\@@fc@numstr\ }%
   \fi
   \let\@@fc@numstr#2\relax
   \edef#2{\@@fc@numstr\@unitstring{\@strctr}\ \@hundred}%
\fi
\@strctr=#1\relax \@modulo{\@strctr}{100}%
\ifnum#1>100\relax
  \ifnum\@strctr>0\relax
    \let\@@fc@numstr#2\relax
    \edef#2{\@@fc@numstr\ \@andname\ }%
  \fi
\fi
\ifnum\@strctr>19\relax
  \divide\@strctr by 10\relax
  \let\@@fc@numstr#2\relax
  \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
  \@strctr=#1\relax \@modulo{\@strctr}{10}%
  \ifnum\@strctr>0\relax
    \let\@@fc@numstr#2\relax
    \edef#2{\@@fc@numstr-\@unitstring{\@strctr}}%
  \fi
\else
  \ifnum\@strctr<10\relax
    \ifnum\@strctr=0\relax
       \ifnum#1<100\relax
          \let\@@fc@numstr#2\relax
          \edef#2{\@@fc@numstr\@unitstring{\@strctr}}%
       \fi
    \else
      \let\@@fc@numstr#2\relax
      \edef#2{\@@fc@numstr\@unitstring{\@strctr}}%
    \fi
  \else
    \@modulo{\@strctr}{10}%
    \let\@@fc@numstr#2\relax
    \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
  \fi
\fi
}%
```

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

```
\DeclareRobustCommand{\@numberstringMenglish}[2]{%
  \let\@unitstring=\@@unitstringenglish
  \let\@teenstring=\@@teenstringenglish
  \let\@tenstring=\@@tenstringenglish
  \def\@hundred{hundred}\def\@thousand{thousand}%
  \def\@andname{and}%
  \@@numberstringenglish{#1}{#2}%
}%
\global\let\@numberstringMenglish\@numberstringMenglish
```

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

```
\global\let\@numberstringFenglish=\@numberstringMenglish
\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.

```
\gdef\@NumberstringMenglish#1#2{%
  \let\@unitstring=\@@Unitstringenglish
  \let\@teenstring=\@@Teenstringenglish
  \let\@tenstring=\@@Tenstringenglish
  \def\@hundred{Hundred}\def\@thousand{Thousand}%
  \def\@andname{and}%
  \@@numberstringenglish{#1}{#2}%
}%
```

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

```
\global\let\@NumberstringFenglish=\@NumberstringMenglish
\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:

```
\gdef\@@unitthstringenglish#1{%
  \ifcase#1\relax
    zeroth%
    \or first%
    \or second%
    \or third%
    \or fourth%
    \or fifth%
    \or sixth%
    \or seventh%
    \or eighth%
    \or ninth%
  \fi
}%
```

Next the tens:

```
\gdef\@@tenthstringenglish#1{%
  \ifcase#1\relax
    \or tenth%
    \or twentieth%
    \or thirtieth%
    \or fortieth%
    \or fiftieth%
    \or sixtieth%
    \or seventieth%
    \or eightieth%
    \or ninetieth%
  \fi
}%
```

The teens:

```
\gdef\@@teenthstringenglish#1{%
  \ifcase#1\relax
    tenth%
    \or eleventh%
    \or twelfth%
    \or thirteenth%
    \or fourteenth%
    \or fifteenth%
    \or sixteenth%
    \or seventeenth%
    \or eighteenth%
    \or nineteenth%
  \fi
}%
```

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

```
\gdef\@@Unitthstringenglish#1{%
  \ifcase#1\relax
    Zeroth%
    \or First%
    \or Second%
    \or Third%
    \or Fourth%
    \or Fifth%
    \or Sixth%
    \or Seventh%
    \or Eighth%
    \or Ninth%
  \fi
}%
```

The tens:

```
\gdef\@@Tenthstringenglish#1{%
  \ifcase#1\relax
    \or Tenth%
    \or Twentieth%
    \or Thirtieth%
    \or Fortieth%
    \or Fiftieth%
    \or Sixtieth%
    \or Seventieth%
    \or Eightieth%
    \or Ninetieth%
  \fi
}%
```

The teens:

```
\gdef\@@Teenthstringenglish#1{%
  \ifcase#1\relax
    Tenth%
    \or Eleventh%
    \or Twelfth%
    \or Thirteenth%
    \or Fourteenth%
    \or Fifteenth%
    \or Sixteenth%
    \or Seventeenth%
    \or Eighteenth%
    \or Nineteenth%
  \fi
}%
```

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.

```
\gdef\@@ordinalstringenglish#1#2{%
\@strctr=#1\relax
\ifnum#1>99999
\PackageError{fmtcount}{Out of range}%
{This macro only works for values less than 100000 (value given: \number\@strctr)}%
\else
\ifnum#1<0
\PackageError{fmtcount}{Negative numbers not permitted}%
{This macro does not work for negative numbers, however
you can try typing "minus" first, and then pass the modulus of
this number}%
\fi
\def#2{}%
\fi
\@strctr=#1\relax \divide\@strctr by 1000\relax
\ifnum\@strctr>9\relax
```

#1 is greater or equal to 10000

```
  \divide\@strctr by 10
  \ifnum\@strctr>1\relax
    \let\@@fc@ordstr#2\relax
    \edef#2{\@@fc@ordstr\@tenstring{\@strctr}}%
    \@strctr=#1\relax
    \divide\@strctr by 1000\relax
    \@modulo{\@strctr}{10}%
    \ifnum\@strctr>0\relax
      \let\@@fc@ordstr#2\relax
      \edef#2{\@@fc@ordstr-\@unitstring{\@strctr}}%
    \fi
  \else
    \@strctr=#1\relax \divide\@strctr by 1000\relax
    \@modulo{\@strctr}{10}%
    \let\@@fc@ordstr#2\relax
    \edef#2{\@@fc@ordstr\@teenstring{\@strctr}}%
  \fi
  \@strctr=#1\relax \@modulo{\@strctr}{1000}%
  \ifnum\@strctr=0\relax
    \let\@@fc@ordstr#2\relax
    \edef#2{\@@fc@ordstr\ \@thousandth}%
  \else
    \let\@@fc@ordstr#2\relax
    \edef#2{\@@fc@ordstr\ \@thousand}%
  \fi
\else
  \ifnum\@strctr>0\relax
    \let\@@fc@ordstr#2\relax
    \edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
    \@strctr=#1\relax \@modulo{\@strctr}{1000}%
    \let\@@fc@ordstr#2\relax
    \ifnum\@strctr=0\relax
      \edef#2{\@@fc@ordstr\ \@thousandth}%
    \else
      \edef#2{\@@fc@ordstr\ \@thousand}%
    \fi
  \fi
\fi
\@strctr=#1\relax \@modulo{\@strctr}{1000}%
\divide\@strctr by 100
\ifnum\@strctr>0\relax
  \ifnum#1>1000\relax
    \let\@@fc@ordstr#2\relax
    \edef#2{\@@fc@ordstr\ }%
  \fi
  \let\@@fc@ordstr#2\relax
  \edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
  \@strctr=#1\relax \@modulo{\@strctr}{100}%
  \let\@@fc@ordstr#2\relax
  \ifnum\@strctr=0\relax
    \edef#2{\@@fc@ordstr\ \@hundredth}%
  \else
    \edef#2{\@@fc@ordstr\ \@hundred}%
  \fi
\fi
\@strctr=#1\relax \@modulo{\@strctr}{100}%
\ifnum#1>100\relax
  \ifnum\@strctr>0\relax
    \let\@@fc@ordstr#2\relax
    \edef#2{\@@fc@ordstr\ \@andname\ }%
  \fi
\fi
\ifnum\@strctr>19\relax
  \@tmpstrctr=\@strctr
  \divide\@strctr by 10\relax
  \@modulo{\@tmpstrctr}{10}%
  \let\@@fc@ordstr#2\relax
  \ifnum\@tmpstrctr=0\relax
    \edef#2{\@@fc@ordstr\@tenthstring{\@strctr}}%
  \else
    \edef#2{\@@fc@ordstr\@tenstring{\@strctr}}%
  \fi
  \@strctr=#1\relax \@modulo{\@strctr}{10}%
  \ifnum\@strctr>0\relax
    \let\@@fc@ordstr#2\relax
    \edef#2{\@@fc@ordstr-\@unitthstring{\@strctr}}%
  \fi
\else
  \ifnum\@strctr<10\relax
    \ifnum\@strctr=0\relax
      \ifnum#1<100\relax
        \let\@@fc@ordstr#2\relax
        \edef#2{\@@fc@ordstr\@unitthstring{\@strctr}}%
      \fi
    \else
      \let\@@fc@ordstr#2\relax
      \edef#2{\@@fc@ordstr\@unitthstring{\@strctr}}%
    \fi
  \else
    \@modulo{\@strctr}{10}%
    \let\@@fc@ordstr#2\relax
    \edef#2{\@@fc@ordstr\@teenthstring{\@strctr}}%
  \fi
\fi
}%
```

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

```
\DeclareRobustCommand{\@ordinalstringMenglish}[2]{%
  \let\@unitthstring=\@@unitthstringenglish
  \let\@teenthstring=\@@teenthstringenglish
  \let\@tenthstring=\@@tenthstringenglish
  \let\@unitstring=\@@unitstringenglish
  \let\@teenstring=\@@teenstringenglish
  \let\@tenstring=\@@tenstringenglish
  \def\@andname{and}%
  \def\@hundred{hundred}\def\@thousand{thousand}%
  \def\@hundredth{hundredth}\def\@thousandth{thousandth}%
  \@@ordinalstringenglish{#1}{#2}%
}%
\global\let\@ordinalstringMenglish\@ordinalstringMenglish
```

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

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

First letter of each word in upper case:

```
\DeclareRobustCommand{\@OrdinalstringMenglish}[2]{%
  \let\@unitthstring=\@@Unitthstringenglish
  \let\@teenthstring=\@@Teenthstringenglish
  \let\@tenthstring=\@@Tenthstringenglish
  \let\@unitstring=\@@Unitstringenglish
  \let\@teenstring=\@@Teenstringenglish
  \let\@tenstring=\@@Tenstringenglish
  \def\@andname{and}%
  \def\@hundred{Hundred}\def\@thousand{Thousand}%
  \def\@hundredth{Hundredth}\def\@thousandth{Thousandth}%
  \@@ordinalstringenglish{#1}{#2}%
}%
\global\let\@OrdinalstringMenglish\@OrdinalstringMenglish
```

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

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

9.4.4 fc-francais.def

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

Set `francais` to be equivalent to `french`.

```
\global\let\@ordinalMfrancais=\@ordinalMfrench
\global\let\@ordinalFfrancais=\@ordinalFfrench
\global\let\@ordinalNfrancais=\@ordinalNfrench
\global\let\@numberstringMfrancais=\@numberstringMfrench
\global\let\@numberstringFfrancais=\@numberstringFfrench
\global\let\@numberstringNfrancais=\@numberstringNfrench
\global\let\@NumberstringMfrancais=\@NumberstringMfrench
\global\let\@NumberstringFfrancais=\@NumberstringFfrench
\global\let\@NumberstringNfrancais=\@NumberstringNfrench
\global\let\@ordinalstringMfrancais=\@ordinalstringMfrench
\global\let\@ordinalstringFfrancais=\@ordinalstringFfrench
\global\let\@ordinalstringNfrancais=\@ordinalstringNfrench
\global\let\@OrdinalstringMfrancais=\@OrdinalstringMfrench
\global\let\@OrdinalstringFfrancais=\@OrdinalstringFfrench
\global\let\@OrdinalstringNfrancais=\@OrdinalstringNfrench
```

9.4.5 fc-french.def

Definitions for French.

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

Package fcprefix is needed to format the prefix ⟨*n*⟩ in ⟨*n*⟩illion or ⟨*n*⟩illiard. Big numbers were developped based 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`'.

```
\def\fc@french@set@plural#1#2#3#4#5#6{%
  \ifthenelse{\equal{#2}{reformed}}{%
    \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{#3}%
  }{%
    \ifthenelse{\equal{#2}{traditional}}{%
      \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{#4}%
    }{%
      \ifthenelse{\equal{#2}{reformed o}}{%
        \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{#5}%
      }{%
        \ifthenelse{\equal{#2}{traditional o}}{%
          \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{#6}%
        }{%
          \ifthenelse{\equal{#2}{always}}{%
            \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{0}%
          }{%
            \ifthenelse{\equal{#2}{never}}{%
              \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{1}%
            }{%
              \ifthenelse{\equal{#2}{multiple}}{%
                \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{2}%
              }{%
                \ifthenelse{\equal{#2}{multiple g-last}}{%
                  \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{3}%
                }{%
                  \ifthenelse{\equal{#2}{multiple l-last}}{%
                    \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{4}%
                  }{%
                    \ifthenelse{\equal{#2}{multiple lng-last}}{%
                      \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{5}%
                    }{%
                      \ifthenelse{\equal{#2}{multiple ng-last}}{%
                        \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{6}%
                      }{%
                        \PackageError{fmtcount}{Unexpected argument}{%
                          `#2' was unexpected: french option `#1 plural' expects `reformed'
```

```
                            ‘reformed o’, ‘traditional o’, ‘always’, ‘never’, ‘multiple’, ‘mul
                            ‘multiple l-last’, ‘multiple lng-last’, or ‘multiple ng-last’.%
                          }}}}}}}}}}}}
```

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`’.

```
\def\@tempa#1#2#3{%
  \define@key{fcfrench}{#1 plural}[reformed]{%
    \fc@french@set@plural{#1}{##1}{#2}{#2}{#3}{#3}%
  }%
}
\@tempa{vingt}{4}{5}
\@tempa{cent}{4}{5}
\@tempa{mil}{0}{0}
\@tempa{n-illion}{2}{6}
\@tempa{n-illiard}{2}{6}
```

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

```
\define@key{fcfrench}{all plural}[reformed]{%
  \csname KV@fcfrench@vingt plural\endcsname{#1}%
  \csname KV@fcfrench@cent plural\endcsname{#1}%
  \csname KV@fcfrench@mil plural\endcsname{#1}%
  \csname KV@fcfrench@n-illion plural\endcsname{#1}%
  \csname KV@fcfrench@n-illiard plural\endcsname{#1}%
}
```

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

```
\define@key{fcfrench}{dash or space}[reformed]{%
  \ifthenelse{\equal{#1}{traditional}}{%
    \let\fc@frenchoptions@supermillion@dos\space%
    \let\fc@frenchoptions@submillion@dos\space
  }{%
    \ifthenelse{\equal{#1}{reformed}\or\equal{#1}{1990}}{%
      \let\fc@frenchoptions@supermillion@dos\space
      \def\fc@frenchoptions@submillion@dos{-}%
    }{%
      \ifthenelse{\equal{#1}{always}}{%
        \def\fc@frenchoptions@supermillion@dos{-}%
        \def\fc@frenchoptions@submillion@dos{-}%
      }{%
        \PackageError{fmtcount}{Unexpected argument}{%
```

```
          French option ‘dash or space’ expects ‘always’, ‘reformed’ or ‘traditional’
        }
      }%
    }%
  }%
}
```

Option ‘`scale`’, can take 3 possible values:

`long`	for which ⟨n⟩illions & ⟨n⟩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 ⟨n⟩illions only are used with $10^{3\times n+3} = 1\langle n\rangle$illion
`recursive`	for which 10^{18} = un milliard de milliards

```
\define@key{fcfrench}{scale}[recursive]{%
  \ifthenelse{\equal{#1}{long}}{%
      \let\fc@poweroften\fc@@pot@longscalefrench
  }{%
    \ifthenelse{\equal{#1}{recursive}}{%
      \let\fc@poweroften\fc@@pot@recursivefrench
    }{%
      \ifthenelse{\equal{#1}{short}}{%
        \let\fc@poweroften\fc@@pot@shortscalefrench
      }{%
        \PackageError{fmtcount}{Unexpected argument}{%
          French option ‘scale’ expects ‘long’, ‘recursive’ or ‘short’
        }
      }%
    }%
  }%
}
```

Option ‘`n-illiard upto`’ is ignored if ‘`scale`’ is different from ‘`long`’. It can take the following values:

`infinity`	in that case ⟨n⟩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 ⟨n⟩illions”

```
\define@key{fcfrench}{n-illiard upto}[infinity]{%
  \ifthenelse{\equal{#1}{infinity}}{%
      \def\fc@longscale@nilliard@upto{0}%
  }{%
    \ifthenelse{\equal{#1}{infty}}{%
      \def\fc@longscale@nilliard@upto{0}%
    }{%
      \if Q\ifnum9<1#1Q\fi\else
      \PackageError{fmtcount}{Unexpected argument}{%
        French option ‘milliard threshold’ expects ‘infinity’, or equivalently ‘infty’, or a
        integer.}%
      \fi
      \def\fc@longscale@nilliard@upto{#1}%
    }}%
}
```

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.

```
\def\@tempa#1{%
  \define@key{fcfrench}{#1}[]{%
    \PackageError{fmtcount}{Unexpected argument}{French option with key ‘#1’ does not take
      any value}}%
  \expandafter\def\csname KV@fcfrench@#1@default\endcsname{%
    \def\fmtcount@french{#1}}%
}%
\@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.

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

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

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

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

\def\fc@CaseIden#1\@nil{%
  #1%
}
\def\fc@UpperCaseAll#1\@nil{%
  \uppercase{#1}%
}

\let\fc@case\fc@CaseIden

```

\@ ordinalMfrench

```
\newcommand*{\@ordinalMfrench}[2]{%
\iffmtord@abbrv
  \edef#2{\number#1\relax\noexpand\fmtord{e}}%
\else
  \ifnum#1=1\relax
    \edef#2{\number#1\relax\noexpand\fmtord{er}}%
  \else
    \edef#2{\number#1\relax\noexpand\fmtord{eme}}%
  \fi
\fi}
```

\@ ordinalFfrench

```
\newcommand*{\@ordinalFfrench}[2]{%
\iffmtord@abbrv
  \edef#2{\number#1\relax\noexpand\fmtord{e}}%
\else
  \ifnum#1=1 %
     \edef#2{\number#1\relax\noexpand\fmtord{i\`ere}}%
  \else
     \edef#2{\number#1\relax\noexpand\fmtord{i\`eme}}%
  \fi
\fi}
```

In French neutral gender and masculine gender are formally identical.

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

\@ @unitstringfrench

```
\newcommand*{\@@unitstringfrench}[1]{%
\noexpand\fc@case
\ifcase#1 %
z\'ero%
\or un%
\or deux%
\or trois%
\or quatre%
\or cinq%
\or six%
\or sept%
\or huit%
\or neuf%
\fi
\noexpand\@nil
}
```

\@ @tenstringfrench

```
\newcommand*{\@@tenstringfrench}[1]{%
\noexpand\fc@case
\ifcase#1 %
\or dix%
\or vingt%
\or trente%
\or quarante%
\or cinquante%
\or soixante%
\or septante%
\or huitante%
\or nonante%
\or cent%
\fi
\noexpand\@nil
}
```

\@ @teenstringfrench

```
\newcommand*{\@@teenstringfrench}[1]{%
\noexpand\fc@case
\ifcase#1 %
    dix%
\or onze%
\or douze%
\or treize%
\or quatorze%
\or quinze%
\or seize%
\or dix\noexpand\@nil-\noexpand\fc@case sept%
\or dix\noexpand\@nil-\noexpand\fc@case huit%
\or dix\noexpand\@nil-\noexpand\fc@case neuf%
\fi
\noexpand\@nil
}
```

\@ @seventiesfrench

```
\newcommand*{\@@seventiesfrench}[1]{%
\@tenstring{6}%
\ifnum#1=1 %
\fc@frenchoptions@submillion@dos\@andname\fc@frenchoptions@submillion@dos
\else
-%
\fi
\@teenstring{#1}%
}
```

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

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

Implicit arguments as:

`\count0`	weight w of the number $d_{w+1}d_w$ to be formatted
`\count1`	same as `\#1`
`\count6`	input, counter giving the least weight of non zero digits in top level formatted number integral part, with rounding down to a multiple of 3,
`\count9`	input, counter giving the power type of the power of ten following the eighties to be formatted; that is '1' for "mil" and '2' for "⟨n⟩illion|⟨n⟩illiard".

```
\newcommand*\@@eightiesfrench[1]{%
\fc@case quatre\@nil-\noexpand\fc@case vingt%
\ifnum#1>0 %
  \ifnum\fc@frenchoptions@vingt@plural=0 % vingt plural=always
  s%
  \fi
  \noexpand\@nil
  -\@unitstring{#1}%
\else
  \ifcase\fc@frenchoptions@vingt@plural\space
    s% 0: always
  \or
    % 1: never
  \or
    s% 2: multiple
  \or
    % 3: multiple g-last
    \ifnum\count0=\count6\ifnum\count9=0 s\fi\fi
  \or
    % 4: multiple l-last
    \ifnum\count9=1 %
    \else
      s%
    \fi
  \or
    % 5: multiple lng-last
    \ifnum\count9=1 %
    \else
      \ifnum\count0>0 %
        s%
      \fi
    \fi
  \or
    % or 6: multiple ng-last
    \ifnum\count0>0 %
      s%
    \fi
  \fi
  \noexpand\@nil
\fi
}
\newcommand*{\@@ninetiesfrench}[1]{%
\fc@case quatre\@nil-\noexpand\fc@case vingt%
\ifnum\fc@frenchoptions@vingt@plural=0 % vingt plural=always
  s%
\fi
\noexpand\@nil
-\@teenstring{#1}%
}
```

```
\newcommand*{\@@seventiesfrenchswiss}[1]{%
\@tenstring{7}%
\ifnum#1=1\ \@andname\ \fi
\ifnum#1>1-\fi
\ifnum#1>0 \@unitstring{#1}\fi
}
\newcommand*{\@@eightiesfrenchswiss}[1]{%
\@tenstring{8}%
\ifnum#1=1\ \@andname\ \fi
\ifnum#1>1-\fi
\ifnum#1>0 \@unitstring{#1}\fi
}
\newcommand*{\@@ninetiesfrenchswiss}[1]{%
\@tenstring{9}%
\ifnum#1=1\ \@andname\ \fi
\ifnum#1>1-\fi
\ifnum#1>0 \@unitstring{#1}\fi
}
```

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

```
\newcommand*\fc@french@common{%
  \let\@unitstring=\@@unitstringfrench
  \let\@teenstring=\@@teenstringfrench
  \let\@tenstring=\@@tenstringfrench
  \def\@hundred{cent}%
  \def\@andname{et}%
}
```

```
\DeclareRobustCommand{\@numberstringMfrenchswiss}[2]{%
\let\fc@case\fc@CaseIden
\fc@french@common
\let\@seventies=\@@seventiesfrenchswiss
\let\@eighties=\@@eightiesfrenchswiss
\let\@nineties=\@@ninetiesfrenchswiss
\let\fc@nbrstr@preamble\@empty
\let\fc@nbrstr@postamble\@empty
\@@numberstringfrench{#1}{#2}}
\DeclareRobustCommand{\@numberstringMfrenchfrance}[2]{%
\let\fc@case\fc@CaseIden
\fc@french@common
\let\@seventies=\@@seventiesfrench
\let\@eighties=\@@eightiesfrench
\let\@nineties=\@@ninetiesfrench
\let\fc@nbrstr@preamble\@empty
\let\fc@nbrstr@postamble\@empty
\@@numberstringfrench{#1}{#2}}
\DeclareRobustCommand{\@numberstringMfrenchbelgian}[2]{%
\let\fc@case\fc@CaseIden
\fc@french@common
```

```
\let\@seventies=\@@seventiesfrenchswiss
\let\@eighties=\@@eightiesfrench
\let\@nineties=\@@ninetiesfrench
\let\fc@nbrstr@preamble\@empty
\let\fc@nbrstr@postamble\@empty
\@@numberstringfrench{#1}{#2}}
\let\@numberstringMfrench=\@numberstringMfrenchfrance
\DeclareRobustCommand{\@numberstringFfrenchswiss}[2]{%
\let\fc@case\fc@CaseIden
\fc@french@common
\let\@seventies=\@@seventiesfrenchswiss
\let\@eighties=\@@eightiesfrenchswiss
\let\@nineties=\@@ninetiesfrenchswiss
\let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
\let\fc@nbrstr@postamble\@empty
\@@numberstringfrench{#1}{#2}}
\DeclareRobustCommand{\@numberstringFfrenchfrance}[2]{%
\let\fc@case\fc@CaseIden
\fc@french@common
\let\@seventies=\@@seventiesfrench
\let\@eighties=\@@eightiesfrench
\let\@nineties=\@@ninetiesfrench
\let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
\let\fc@nbrstr@postamble\@empty
\@@numberstringfrench{#1}{#2}}
\DeclareRobustCommand{\@numberstringFfrenchbelgian}[2]{%
\let\fc@case\fc@CaseIden
\fc@french@common
\let\@seventies=\@@seventiesfrenchswiss
\let\@eighties=\@@eightiesfrench
\let\@nineties=\@@ninetiesfrench
\let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
\let\fc@nbrstr@postamble\@empty
\@@numberstringfrench{#1}{#2}}
\let\@numberstringFfrench=\@numberstringFfrenchfrance
\let\@ordinalstringNfrench\@ordinalstringMfrench
\DeclareRobustCommand{\@NumberstringMfrenchswiss}[2]{%
\let\fc@case\fc@UpperCaseFirstLetter
\fc@french@common
\let\@seventies=\@@seventiesfrenchswiss
\let\@eighties=\@@eightiesfrenchswiss
\let\@nineties=\@@ninetiesfrenchswiss
\let\fc@nbrstr@preamble\@empty
\let\fc@nbrstr@postamble\@empty
\@@numberstringfrench{#1}{#2}}
\DeclareRobustCommand{\@NumberstringMfrenchfrance}[2]{%
\let\fc@case\fc@UpperCaseFirstLetter
\fc@french@common
\let\@seventies=\@@seventiesfrench
```

```
\let\@eighties=\@@eightiesfrench
\let\@nineties=\@@ninetiesfrench
\let\fc@nbrstr@preamble\@empty
\let\fc@nbrstr@postamble\@empty
\@@numberstringfrench{#1}{#2}}
\DeclareRobustCommand{\@NumberstringMfrenchbelgian}[2]{%
\let\fc@case\fc@UpperCaseFirstLetter
\fc@french@common
\let\@seventies=\@@seventiesfrenchswiss
\let\@eighties=\@@eightiesfrench
\let\@nineties=\@@ninetiesfrench
\let\fc@nbrstr@preamble\@empty
\let\fc@nbrstr@postamble\@empty
\@@numberstringfrench{#1}{#2}}
\let\@NumberstringMfrench=\@NumberstringMfrenchfrance
\DeclareRobustCommand{\@NumberstringFfrenchswiss}[2]{%
\let\fc@case\fc@UpperCaseFirstLetter
\fc@french@common
\let\@seventies=\@@seventiesfrenchswiss
\let\@eighties=\@@eightiesfrenchswiss
\let\@nineties=\@@ninetiesfrenchswiss
\let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
\let\fc@nbrstr@postamble\@empty
\@@numberstringfrench{#1}{#2}}
\DeclareRobustCommand{\@NumberstringFfrenchfrance}[2]{%
\let\fc@case\fc@UpperCaseFirstLetter
\fc@french@common
\let\@seventies=\@@seventiesfrench
\let\@eighties=\@@eightiesfrench
\let\@nineties=\@@ninetiesfrench
\let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
\let\fc@nbrstr@postamble\@empty
\@@numberstringfrench{#1}{#2}}
\DeclareRobustCommand{\@NumberstringFfrenchbelgian}[2]{%
\let\fc@case\fc@UpperCaseFirstLetter
\fc@french@common
\let\@seventies=\@@seventiesfrenchswiss
\let\@eighties=\@@eightiesfrench
\let\@nineties=\@@ninetiesfrench
\let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
\let\fc@nbrstr@postamble\@empty
\@@numberstringfrench{#1}{#2}}
\let\@NumberstringFfrench=\@NumberstringFfrenchfrance
\let\@NumberstringNfrench\@NumberstringMfrench
\DeclareRobustCommand{\@ordinalstringMfrenchswiss}[2]{%
\let\fc@case\fc@CaseIden
\let\fc@first=\fc@@firstfrench
\fc@french@common
\let\@seventies=\@@seventiesfrenchswiss
```

```
\let\@eighties=\@@eightiesfrenchswiss
\let\@nineties=\@@ninetiesfrenchswiss
\@@ordinalstringfrench{#1}{#2}%
}
\newcommand*\fc@@firstfrench{premier}
\newcommand*\fc@@firstFfrench{premi\`ere}
\DeclareRobustCommand{\@ordinalstringMfrenchfrance}[2]{%
\let\fc@case\fc@CaseIden
\let\fc@first=\fc@@firstfrench
\fc@french@common
\let\@seventies=\@@seventiesfrench
\let\@eighties=\@@eightiesfrench
\let\@nineties=\@@ninetiesfrench
\@@ordinalstringfrench{#1}{#2}}
\DeclareRobustCommand{\@ordinalstringMfrenchbelgian}[2]{%
\let\fc@case\fc@CaseIden
\let\fc@first=\fc@@firstfrench
\fc@french@common
\let\@seventies=\@@seventiesfrench
\let\@eighties=\@@eightiesfrench
\let\@nineties=\@@ninetiesfrench
\@@ordinalstringfrench{#1}{#2}%
}
\let\@ordinalstringMfrench=\@ordinalstringMfrenchfrance
\DeclareRobustCommand{\@ordinalstringFfrenchswiss}[2]{%
\let\fc@case\fc@CaseIden
\let\fc@first=\fc@@firstFfrench
\fc@french@common
\let\@seventies=\@@seventiesfrenchswiss
\let\@eighties=\@@eightiesfrenchswiss
\let\@nineties=\@@ninetiesfrenchswiss
\@@ordinalstringfrench{#1}{#2}%
}
\DeclareRobustCommand{\@ordinalstringFfrenchfrance}[2]{%
\let\fc@case\fc@CaseIden
\let\fc@first=\fc@@firstFfrench
\fc@french@common
\let\@seventies=\@@seventiesfrench
\let\@eighties=\@@eightiesfrench
\let\@nineties=\@@ninetiesfrench
\@@ordinalstringfrench{#1}{#2}%
}
\DeclareRobustCommand{\@ordinalstringFfrenchbelgian}[2]{%
\let\fc@case\fc@CaseIden
\let\fc@first=\fc@@firstFfrench
\fc@french@common
\let\@seventies=\@@seventiesfrench
\let\@eighties=\@@eightiesfrench
\let\@nineties=\@@ninetiesfrench
```

```
\@@ordinalstringfrench{#1}{#2}%
}
\let\@ordinalstringFfrench=\@ordinalstringFfrenchfrance
\let\@ordinalstringNfrench\@ordinalstringMfrench
\DeclareRobustCommand{\@OrdinalstringMfrenchswiss}[2]{%
\let\fc@case\fc@UpperCaseFirstLetter
\let\fc@first=\fc@@firstfrench
\fc@french@common
\let\@seventies=\@@seventiesfrenchswiss
\let\@eighties=\@@eightiesfrenchswiss
\let\@nineties=\@@ninetiesfrenchswiss
\@@ordinalstringfrench{#1}{#2}%
}
\DeclareRobustCommand{\@OrdinalstringMfrenchfrance}[2]{%
\let\fc@case\fc@UpperCaseFirstLetter
\let\fc@first=\fc@@firstfrench
\fc@french@common
\let\@seventies=\@@seventiesfrench
\let\@eighties=\@@eightiesfrench
\let\@nineties=\@@ninetiesfrench
\@@ordinalstringfrench{#1}{#2}%
}
\DeclareRobustCommand{\@OrdinalstringMfrenchbelgian}[2]{%
\let\fc@case\fc@UpperCaseFirstLetter
\let\fc@first=\fc@@firstfrench
\fc@french@common
\let\@seventies=\@@seventiesfrench
\let\@eighties=\@@eightiesfrench
\let\@nineties=\@@ninetiesfrench
\@@ordinalstringfrench{#1}{#2}%
}
\let\@OrdinalstringMfrench=\@OrdinalstringMfrenchfrance
\DeclareRobustCommand{\@OrdinalstringFfrenchswiss}[2]{%
\let\fc@case\fc@UpperCaseFirstLetter
\let\fc@first=\fc@@firstfrench
\fc@french@common
\let\@seventies=\@@seventiesfrenchswiss
\let\@eighties=\@@eightiesfrenchswiss
\let\@nineties=\@@ninetiesfrenchswiss
\@@ordinalstringfrench{#1}{#2}%
}
\DeclareRobustCommand{\@OrdinalstringFfrenchfrance}[2]{%
\let\fc@case\fc@UpperCaseFirstLetter
\let\fc@first=\fc@@firstFfrench
\fc@french@common
\let\@seventies=\@@seventiesfrench
\let\@eighties=\@@eightiesfrench
\let\@nineties=\@@ninetiesfrench
\@@ordinalstringfrench{#1}{#2}%
```

```
}
\DeclareRobustCommand{\@OrdinalstringFfrenchbelgian}[2]{%
\let\fc@case\fc@UpperCaseFirstLetter
\let\fc@first=\fc@@firstFfrench
\fc@french@common
\let\@seventies=\@@seventiesfrench
\let\@eighties=\@@eightiesfrench
\let\@nineties=\@@ninetiesfrench
\@@ordinalstringfrench{#1}{#2}%
}
\let\@OrdinalstringFfrench=\@OrdinalstringFfrenchfrance
\let\@OrdinalstringNfrench\@OrdinalstringMfrench
```

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

```
\@ifundefined{fc@@do@plural@mark}{}{\PackageError{fmtcount}{Duplicate definition}{Redefinit
    ‘fc@@do@plural@mark’}}
```

Arguments as follows:

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

Implicit arguments as follows:

`\count0`	input, counter giving the weight w, this is expected to be multiple of 3,
`\count1`	input, counter giving the plural value of multiplied object ⟨*n*⟩illiard, ⟨*n*⟩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.

```
\def\fc@@do@plural@mark#1{%
  \ifcase\count10 %
    #1% 0=always
   \or% 1=never
   \or% 2=multiple
     \ifnum\count1>1 %
       #1%
     \fi
  \or% 3= multiple g-last
     \ifnum\count1>1 %
       \ifnum\count0=\count6 %
         #1%
       \fi
     \fi
  \or% 4= multiple l-last
```

```
    \ifnum\count1>1 %
      \ifnum\count9=1 %
      \else
        #1%
      \fi
    \fi
  \or% 5= multiple lng-last
    \ifnum\count1>1 %
      \ifnum\count9=1 %
      \else
        \if\count0>\count6 %
          #1%
        \fi
      \fi
    \fi
  \or% 6= multiple ng-last
    \ifnum\count1>1 %
      \ifnum\count0>\count6 %
        #1%
      \fi
    \fi
  \fi
}
```

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

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

`\fc@@nbrstr@Fpreamble`

```
\def\fc@@nbrstr@Fpreamble{%
  \fc@read@unit{\count1}{0}%
  \ifnum\count1=1 %
      \let\fc@case@save\fc@case
      \def\fc@case{\noexpand\fc@case}%
      \def\@nil{\noexpand\@nil}%
     \let\fc@nbrstr@postamble\fc@@nbrstr@Fpostamble
  \fi
}
```

`\fc@@nbrstr@Fpostamble`

```
\def\fc@@nbrstr@Fpostamble{%
  \let\fc@case\fc@case@save
  \expandafter\fc@get@last@word\expandafter{\@tempa}\@tempb\@tempc
  \def\@tempd{un}%
  \ifx\@tempc\@tempd
    \let\@tempc\@tempa
    \edef\@tempa{\@tempb\fc@case une\@nil}%
  \fi
}
```

`\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.

```
\@ifundefined{fc@@pot@longscalefrench}{}{%
  \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
    ‘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

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

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

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

Let `\count1` be the plural value.

```
    \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.

```
    \count2\count0 %
    \divide\count2 by 6 %
    \count3\count2 %
    \multiply\count3 by 6 %
    \count3-\count3 %
    \advance\count3 by \count0 %
    \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)".

```
      \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` `\@temph` to '1' for "mil(le)", and to '2' otherwise.

```
        \edef\@temph{%
          \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.

```
          1%
        \else
          \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`.

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

Here option '`n-illiard upto`' is '`infinity`', so we always use "$\langle n \rangle$illiard(s)".

```
              2%
            \else
```

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

```
              \ifnum\count2>\fc@longscale@nilliard@upto
                1%
              \else
                2%
              \fi
            \fi
          \else
            2%
          \fi
        \fi
      }%
      \ifnum\@temph=1 %
```

Here 10^w is formatted as "mil(le)".

```
        \count10=\fc@frenchoptions@mil@plural\space
        \edef\@tempe{%
          \noexpand\fc@case
           mil%
           \fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark
          \noexpand\@nil
        }%
      \else
        % weight >= 6
        \expandafter\fc@@latin@cardinal@pefix\expandafter{\the\count2}\@tempg
        % now form the xxx-illion(s) or xxx-illiard(s) word
        \ifnum\count3>2 %
           \toks10{illiard}%
           \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space
        \else
           \toks10{illion}%
           \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
        \fi
        \edef\@tempe{%
          \noexpand\fc@case
          \@tempg
          \the\toks10 %
          \fc@@do@plural@mark s%
          \noexpand\@nil
        }%
      \fi
     \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.

```
    \let\@tempe\@empty
    \def\@temph{0}%
   \fi
  \else
```

Case of $w = 0$.

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

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

```
  \expandafter\toks\expandafter1\expandafter{\@tempe}%
  \toks0{#2}%
  \edef\@tempa{\the\toks0 \@temph \def\noexpand#3{\the\toks1}}%
  \expandafter
 }\@tempa
}
```

`\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.

```
\@ifundefined{fc@@pot@shortscalefrench}{}{%
 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
   ‘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

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

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

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

And let `\count1` be the plural value.

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

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

```
  \count2\count0 %
  \divide\count2 by 3 %
  \advance\count2 by -1 %
```

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

```
  \ifnum\count0>0 % If weight>=3, i.e we do have to append thousand or n-illion(s)/n-illi
    \ifnum\count1>0 % we have at least one thousand/n-illion/n-illiard
       \ifnum\count2=0 %
         \def\@temph{1}%
         \count1=\fc@frenchoptions@mil@plural\space
         \edef\@tempe{%
           mil%
           \fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark
         }%
       \else
         \def\@temph{2}%
         % weight >= 6
         \expandafter\fc@@latin@cardinal@pefix\expandafter{\the\count2}\@tempg
         \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
         \edef\@tempe{%
           \noexpand\fc@case
           \@tempg
           illion%
           \fc@@do@plural@mark s%
           \noexpand\@nil
         }%
       \fi
    \else
```

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

```
      \def\@temph{0}%
      \let\@tempe\@empty
    \fi
  \else
```

Here $w = 0$.

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

\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.

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

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

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

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

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

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

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

```
    \ifnum\count1>0 %
      \count2\count0 %
      \divide\count2 by 9 %
      \advance\count2 by -1 %
      \let\@tempe\@empty
      \edef\@tempf{\fc@frenchoptions@supermillion@dos
        de\fc@frenchoptions@supermillion@dos\fc@case milliards\@nil}%
      \count11\count0 %
      \ifnum\count2>0 %
```

```
        \count3\count2 %
        \count3-\count3 %
        \multiply\count3 by 9 %
        \advance\count11 by \count3 %
        \loop
           % (\count2, \count3) <- (\count2 div 2, \count2 mod 2)
           \count3\count2 %
           \divide\count3 by 2 %
           \multiply\count3 by 2 %
           \count3-\count3 %
           \advance\count3 by \count2 %
           \divide\count2 by 2 %
           \ifnum\count3=1 %
             \let\@tempg\@tempe
             \edef\@tempe{\@tempg\@tempf}%
           \fi
           \let\@tempg\@tempf
           \edef\@tempf{\@tempg\@tempg}%
           \ifnum\count2>0 %
        \repeat
      \fi
      \divide\count11 by 3 %
      \ifcase\count11 % 0 .. 5
        % 0 => d milliard(s) (de milliards)*
        \def\@temph{2}%
        \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space
      \or  % 1 => d mille milliard(s) (de milliards)*
        \def\@temph{1}%
        \count10=\fc@frenchoptions@mil@plural\space
      \or % 2 => d million(s) (de milliards)*
        \def\@temph{2}%
        \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
      \or % 3 => d milliard(s) (de milliards)*
        \def\@temph{2}%
        \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space
      \or % 4 => d mille milliards (de milliards)*
        \def\@temph{1}%
        \count10=\fc@frenchoptions@mil@plural\space
      \else % 5 => d million(s) (de milliards)*
        \def\@temph{2}%
        \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
      \fi
      \let\@tempg\@tempe
      \edef\@tempf{%
        \ifcase\count11 % 0 .. 5
        \or
          mil\fc@@do@plural@mark \fc@frenchoptions@mil@plural@mark
        \or
          million\fc@@do@plural@mark s%
        \or
          milliard\fc@@do@plural@mark s%
        \or
          mil\fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark
          \noexpand\@nil\fc@frenchoptions@supermillion@dos
          \noexpand\fc@case milliards% 4
        \or
          million\fc@@do@plural@mark s%
          \noexpand\@nil\fc@frenchoptions@supermillion@dos
          de\fc@frenchoptions@supermillion@dos\noexpand\fc@case  milliards% 5
        \fi
      }%
      \edef\@tempe{%
        \ifx\@tempf\@empty\else
         \expandafter\fc@case\@tempf\@nil
        \fi
        \@tempg
      }%
    \else
       \def\@temph{0}%
       \let\@tempe\@empty
    \fi
```

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

```
      \expandafter\toks\expandafter1\expandafter{\@tempe}%
      \toks0{#2}%
      \edef\@tempa{\the\toks0 \@temph \def\noexpand#3{\the\toks1}}%
      \expandafter
    }\@tempa
}
```

`\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.

```
\@ifundefined{fc@muladdfrench}{}{%
  \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
    ‘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 “metanothing” — for index = 0, “mil(le)” — for index = 1 — or by “$\langle n\rangle$illion(s)|$\langle n\rangle$illiard(s)” — for index = 2

```
\def\fc@muladdfrench#1#2#3{%
  {%
```

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

```
    \edef\@@tempc{#1}%
    \edef\@@tempd{#2}%
    \edef\@tempf{#3}%
    \let\@tempc\@@tempc
    \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”.

```
    \ifnum\@tempc=1 %
      \ifnum\count9=1 %
        \let\@tempd\@empty
      \fi
    \fi
```

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

```
    \edef\@tempg{%
      \@tempd
      \ifx\@tempd\@empty\else
        \ifx\@tempf\@empty\else
          \ifcase\count9 %
          \or
            \fc@frenchoptions@submillion@dos
          \or
             \fc@frenchoptions@supermillion@dos
          \fi
         \fi
       \fi
     \@tempf
    }%
```

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

```
   \edef\@temph{%
     \@tempa
     \ifx\@tempa\@empty\else
       \ifx\@tempg\@empty\else
         \ifcase\count8 %
         \or
           \fc@frenchoptions@submillion@dos
         \or
           \fc@frenchoptions@supermillion@dos
         \fi
       \fi
     \fi
     \@tempg
   }%
```

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

```
    \def\@tempb##1{\def\@tempa{\def\@tempa{##1}}}%
    \expandafter\@tempb\expandafter{\@temph}%
    \expandafter
  }\@tempa
}%
```

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

```
\@ifundefined{fc@lthundredstringfrench}{}{%
  \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
    '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`   intput/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.

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

First save arguments into local temporary macro.

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

Read units d_w to `\count1`.

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

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

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

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

```
  \edef\@tempa{%
    \@tempc
    \ifnum\count2>1 %
      % 20 .. 99
      \ifnum\count2>6 %
        % 70 .. 99
        \ifnum\count2<8 %
           % 70 .. 79
           \@seventies{\count1}%
        \else
          % 80..99
          \ifnum\count2<9 %
            % 80 .. 89
            \@eighties{\count1}%
          \else
            % 90 .. 99
            \@nineties{\count1}%
          \fi
        \fi
      \else
        % 20..69
        \@tenstring{\count2}%
        \ifnum\count1>0 %
           % x1 .. x0
           \ifnum\count1=1 %
             % x1
             \fc@frenchoptions@submillion@dos\@andname\fc@frenchoptions@submillion@dos
           \else
             % x2 .. x9
             -%
           \fi
           \@unitstring{\count1}%
        \fi
      \fi
    \else
      % 0 .. 19
      \ifnum\count2=0 % when tens = 0
        % 0 .. 9
        \ifnum\count1=0 % when units = 0
          % \count3=1 when #1 = 0, i.e. only for the unit of the top level number
          \ifnum\count3=1 %
            \ifnum\fc@max@weight=0 %
              \@unitstring{0}%
            \fi
          \fi
        \else
          % 1 .. 9
          \@unitstring{\count1}%
```

```
            \fi
          \else
            % 10 .. 19
            \@teenstring{\count1}%
          \fi
        \fi
      }%
```

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

```
      \def\@tempb##1{\def\@tempa{\def#1{##1}}}%
      \expandafter\@tempb\expandafter{\@tempa}%
      \expandafter
    }\@tempa
}
```

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

```
\@ifundefined{fc@ltthousandstringfrench}{}{%
  \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
    `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)

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

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

```
      \count4\count0 %
      \advance\count4 by 2 %
      \fc@read@unit{\count2 }{\count4 }%
```

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

```
      \advance\count4 by -1 %
      \count3\count4 %
      \advance\count3 by -1 %
      \fc@check@nonzeros{\count3 }{\count4 }\@tempa
```

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

```
      \edef\@temps{%
        \ifcase\fc@frenchoptions@cent@plural\space
        % 0 => always
        s%
        \or
        % 1 => never
        \or
        % 2 => multiple
        \ifnum\count2>1s\fi
        \or
        % 3 => multiple g-last
          \ifnum\count2>1 \ifnum\@tempa=0 \ifnum\count0=\count6s\fi\fi\fi
        \or
        % 4 => multiple l-last
          \ifnum\count2>1 \ifnum\@tempa=0 \ifnum\count9=0s\else\ifnum\count9=2s\fi\fi\fi\fi
        \fi
      }%
      % compute spacing after cent(s?) into \@tempb
      \expandafter\let\expandafter\@tempb
         \ifnum\@tempa>0 \fc@frenchoptions@submillion@dos\else\@empty\fi
      % now place into \@tempa the hundreds
      \edef\@tempa{%
         \ifnum\count2=0 %
         \else
           \ifnum\count2=1 %
             \expandafter\fc@case\@hundred\@nil
           \else
             \@unitstring{\count2}\fc@frenchoptions@submillion@dos
             \noexpand\fc@case\@hundred\@temps\noexpand\@nil
           \fi
           \@tempb
         \fi
      }%
      % now append to \@tempa the ten and unit
      \fc@lthundredstringfrench\@tempa
```

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

```
    \def\@tempb##1{\def\@tempa{\def#1{##1}}}%
    \expandafter\@tempb\expandafter{\@tempa}%
    \expandafter
  }\@tempa
}
```

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

```
\@ifundefined{@@numberstringfrench}{}{%
  \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

```
\def\@@numberstringfrench#1#2{%
  {%
```

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

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

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

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

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

`\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.

```
\def\fc@@nbrstrfrench@inner{%
```

Now loop, first we compute starting weight as $3\times\left\lfloor\frac{\texttt{\textbackslash fc@max@weight}}{3}\right\rfloor$ into `\count0`.

```
    \count0=\fc@max@weight
    \divide\count0 by 3 %
    \multiply\count0 by 3 %
```

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

```
    \fc@intpart@find@last{\count5 }%
    \count6\count5 %
    \divide\count6 3 %
    \multiply\count6 3 %
    \count8=0 %
    \loop
```

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

```
       \count1\count0 %
       \advance\count1 by 2 %
       \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`.

```
       \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`

```
       \fc@ltthousandstringfrench\@tempd
```

Finally do the multiplication-addition. Implicit arguments are `\@tempa` for input/output growing formatted number, `\count8` for input previous power type, i.e. power type of 10^{w+3}, `\count9` for input current power type, i.e. power type of 10^w.

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

Then iterate.

```
       \count8\count9 %
       \advance\count0 by -3 %
       \ifnum\count6>\count0 \else
    \repeat
}
```

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

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

Arguments are as follows:

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

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

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

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

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

```
    \ifnum\fc@max@weight=0 %
      \ifnum\csname fc@digit@0\endcsname=1 %
        \count0=1 %
      \else
        \count0=0 %
      \fi
    \else
      \count0=0 %
    \fi
    \ifnum\count0=1 %
      \edef\@tempa{\expandafter\fc@case\fc@first\@nil}%
    \else
```

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

```
      \def\@tempa##1{%
        \expandafter\edef\csname fc@frenchoptions@##1@plural\endcsname{%
          \ifcase\csname fc@frenchoptions@##1@plural\endcsname\space
          0% 0: always => always
          \or
          1% 1: never => never
          \or
          6% 2: multiple => multiple  ng-last
          \or
          1% 3: multiple g-last => never
          \or
          5% 4: multiple l-last => multiple lng-last
          \or
          5% 5: multiple lng-last => multiple lng-last
          \or
          6% 6: multiple ng-last => multiple ng-last
          \fi
        }%
      }%
      \@tempa{vingt}%
      \@tempa{cent}%
      \@tempa{mil}%
      \@tempa{n-illion}%
      \@tempa{n-illiard}%
```

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

```
      \let\fc@case@save\fc@case
      \def\fc@case{\noexpand\fc@case}%
      \def\@nil{\noexpand\@nil}%
```

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

```
      \let\@tempa\@empty
      \fc@@nbrstrfrench@inner
```

Now restore `\fc@case`

```
     \let\fc@case\fc@case@save
```

Now we add the “ième” ending

```
      \expandafter\fc@get@last@word\expandafter{\@tempa}\@tempb\@tempc
      \expandafter\fc@get@last@letter\expandafter{\@tempc}\@tempd\@tempe
      \def\@tempf{e}%
      \ifx\@tempe\@tempf
        \edef\@tempa{\@tempb\expandafter\fc@case\@tempd i\`eme\@nil}%
      \else
        \def\@tempf{q}%
        \ifx\@tempe\@tempf
          \edef\@tempa{\@tempb\expandafter\fc@case\@tempd qui\`eme\@nil}%
        \else
          \def\@tempf{f}%
          \ifx\@tempe\@tempf
            \edef\@tempa{\@tempb\expandafter\fc@case\@tempd vi\`eme\@nil}%
          \else
            \edef\@tempa{\@tempb\expandafter\fc@case\@tempc i\`eme\@nil}%
          \fi
        \fi
      \fi
    \fi
```

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

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

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

```
\newcommand*\fc@frenchoptions@setdefaults{%
  \csname KV@fcfrench@all plural\endcsname{reformed}%
  \def\fc@frenchoptions@submillion@dos{-}%
  \let\fc@frenchoptions@supermillion@dos\space
  \let\fc@u@in@duo\@empty% Could be `u'
  % \let\fc@poweroften\fc@@pot@longscalefrench
  \let\fc@poweroften\fc@@pot@recursivefrench
  \def\fc@longscale@nilliard@upto{0}% infinity
  \def\fc@frenchoptions@mil@plural@mark{le}%
}
\fc@frenchoptions@setdefaults
```

9.4.6 fc-frenchb.def

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

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

```
\global\let\@ordinalMfrenchb=\@ordinalMfrench
\global\let\@ordinalFfrenchb=\@ordinalFfrench
\global\let\@ordinalNfrenchb=\@ordinalNfrench
\global\let\@numberstringMfrenchb=\@numberstringMfrench
\global\let\@numberstringFfrenchb=\@numberstringFfrench
```

```
\global\let\@numberstringNfrenchb=\@numberstringNfrench
\global\let\@NumberstringMfrenchb=\@NumberstringMfrench
\global\let\@NumberstringFfrenchb=\@NumberstringFfrench
\global\let\@NumberstringNfrenchb=\@NumberstringNfrench
\global\let\@ordinalstringMfrenchb=\@ordinalstringMfrench
\global\let\@ordinalstringFfrenchb=\@ordinalstringFfrench
\global\let\@ordinalstringNfrenchb=\@ordinalstringNfrench
\global\let\@OrdinalstringMfrenchb=\@OrdinalstringMfrench
\global\let\@OrdinalstringFfrenchb=\@OrdinalstringFfrench
\global\let\@OrdinalstringNfrenchb=\@OrdinalstringNfrench
```

9.4.7 fc-german.def

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

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

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

```
\newcommand{\@ordinalMgerman}[2]{%
  \edef#2{\number#1\relax.}%
}%
\global\let\@ordinalMgerman\@ordinalMgerman
```

Feminine:

```
\newcommand{\@ordinalFgerman}[2]{%
  \edef#2{\number#1\relax.}%
}%
\global\let\@ordinalFgerman\@ordinalFgerman
```

Neuter:

```
\newcommand{\@ordinalNgerman}[2]{%
  \edef#2{\number#1\relax.}%
}%
\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):

```
\gdef\@@unitstringgerman#1{%
  \ifcase#1%
    null%
    \or eins%
    \or zwei%
    \or drei%
    \or vier%
    \or f\"unf%
    \or sechs%
    \or sieben%
    \or acht%
    \or neun%
  \fi
}%
```

Tens (argument must go from 1 to 10):

```
\gdef\@@tenstringgerman#1{%
  \ifcase#1%
    \or zehn%
    \or zwanzig%
    \or drei{\ss}ig%
    \or vierzig%
    \or f\"unfzig%
    \or sechzig%
    \or siebzig%
    \or achtzig%
    \or neunzig%
    \or einhundert%
  \fi
}%
```

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

```
\providecommand*{\einhundert}{einhundert}%
\providecommand*{\eintausend}{eintausend}%
\global\let\einhundert\einhundert
\global\let\eintausend\eintausend
```

Teens:

```
\gdef\@@teenstringgerman#1{%
  \ifcase#1%
    zehn%
    \or elf%
    \or zw\"olf%
    \or dreizehn%
    \or vierzehn%
    \or f\"unfzehn%
    \or sechzehn%
    \or siebzehn%
    \or achtzehn%
    \or neunzehn%
  \fi
}%
```

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

```
\DeclareRobustCommand{\@numberstringMgerman}[2]{%
  \let\@unitstring=\@@unitstringgerman
  \let\@teenstring=\@@teenstringgerman
  \let\@tenstring=\@@tenstringgerman
  \@@numberstringgerman{#1}{#2}%
}%
\global\let\@numberstringMgerman\@numberstringMgerman
```

Feminine and neuter forms:

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

As above, but initial letters in upper case:

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

Feminine and neuter form:

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

As above, but for ordinals.

```
\DeclareRobustCommand{\@ordinalstringMgerman}[2]{%
  \let\@unitthstring=\@@unitthstringMgerman
  \let\@teenthstring=\@@teenthstringMgerman
  \let\@tenthstring=\@@tenthstringMgerman
  \let\@unitstring=\@@unitstringgerman
  \let\@teenstring=\@@teenstringgerman
  \let\@tenstring=\@@tenstringgerman
  \def\@thousandth{tausendster}%
  \def\@hundredth{hundertster}%
  \@@ordinalstringgerman{#1}{#2}%
}%
\global\let\@ordinalstringMgerman\@ordinalstringMgerman
```

Feminine form:

```
\DeclareRobustCommand{\@ordinalstringFgerman}[2]{%
  \let\@unitthstring=\@@unitthstringFgerman
  \let\@teenthstring=\@@teenthstringFgerman
  \let\@tenthstring=\@@tenthstringFgerman
  \let\@unitstring=\@@unitstringgerman
  \let\@teenstring=\@@teenstringgerman
  \let\@tenstring=\@@tenstringgerman
  \def\@thousandth{tausendste}%
  \def\@hundredth{hundertste}%
  \@@ordinalstringgerman{#1}{#2}%
}%
\global\let\@ordinalstringFgerman\@ordinalstringFgerman
```

Neuter form:

```
\DeclareRobustCommand{\@ordinalstringNgerman}[2]{%
  \let\@unitthstring=\@@unitthstringNgerman
  \let\@teenthstring=\@@teenthstringNgerman
  \let\@tenthstring=\@@tenthstringNgerman
  \let\@unitstring=\@@unitstringgerman
  \let\@teenstring=\@@teenstringgerman
  \let\@tenstring=\@@tenstringgerman
  \def\@thousandth{tausendstes}%
  \def\@hundredth{hunderstes}%
  \@@ordinalstringgerman{#1}{#2}%
}%
\global\let\@ordinalstringNgerman\@ordinalstringNgerman
```

As above, but with initial letters in upper case.

```
\DeclareRobustCommand{\@OrdinalstringMgerman}[2]{%
 \@ordinalstringMgerman{#1}{\@@num@str}%
 \edef#2{\noexpand\MakeUppercase\expandonce\@@num@str}%
}%
\global\let\@OrdinalstringMgerman\@OrdinalstringMgerman
```

Feminine form:

```
\DeclareRobustCommand{\@OrdinalstringFgerman}[2]{%
 \@ordinalstringFgerman{#1}{\@@num@str}%
 \edef#2{\noexpand\MakeUppercase\expandonce\@@num@str}%
}%
\global\let\@OrdinalstringFgerman\@OrdinalstringFgerman
```

Neuter form:

```
\DeclareRobustCommand{\@OrdinalstringNgerman}[2]{%
 \@ordinalstringNgerman{#1}{\@@num@str}%
 \edef#2{\noexpand\MakeUppercase\expandonce\@@num@str}%
}%
\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:

```
\gdef\@@unitthstringMgerman#1{%
  \ifcase#1%
    nullter%
    \or erster%
    \or zweiter%
    \or dritter%
    \or vierter%
    \or f\"unfter%
    \or sechster%
    \or siebter%
    \or achter%
    \or neunter%
  \fi
}%
```

Tens:

```
\gdef\@@tenthstringMgerman#1{%
  \ifcase#1%
    \or zehnter%
    \or zwanzigster%
    \or drei{\ss}igster%
    \or vierzigster%
    \or f\"unfzigster%
    \or sechzigster%
    \or siebzigster%
    \or achtzigster%
    \or neunzigster%
  \fi
}%
```

Teens:

```
\gdef\@@teenthstringMgerman#1{%
  \ifcase#1%
    zehnter%
    \or elfter%
    \or zw\"olfter%
    \or dreizehnter%
    \or vierzehnter%
    \or f\"unfzehnter%
    \or sechzehnter%
    \or siebzehnter%
    \or achtzehnter%
    \or neunzehnter%
  \fi
}%
```

Units (feminine):

```
\gdef\@@unitthstringFgerman#1{%
  \ifcase#1%
    nullte%
    \or erste%
    \or zweite%
    \or dritte%
    \or vierte%
    \or f\"unfte%
    \or sechste%
    \or siebte%
    \or achte%
    \or neunte%
  \fi
}%
```

Tens (feminine):

```
\gdef\@@tenthstringFgerman#1{%
  \ifcase#1%
    \or zehnte%
    \or zwanzigste%
    \or drei{\ss}igste%
    \or vierzigste%
    \or f\"unfzigste%
    \or sechzigste%
    \or siebzigste%
    \or achtzigste%
    \or neunzigste%
  \fi
}%
```

Teens (feminine)

```
\gdef\@@teenthstringFgerman#1{%
  \ifcase#1%
    zehnte%
    \or elfte%
    \or zw\"olfte%
    \or dreizehnte%
    \or vierzehnte%
    \or f\"unfzehnte%
    \or sechzehnte%
    \or siebzehnte%
    \or achtzehnte%
    \or neunzehnte%
  \fi
}%
```

Units (neuter):

```
\gdef\@@unitthstringNgerman#1{%
  \ifcase#1%
    nulltes%
    \or erstes%
    \or zweites%
    \or drittes%
    \or viertes%
    \or f\"unftes%
    \or sechstes%
    \or siebtes%
    \or achtes%
    \or neuntes%
  \fi
}%
```

Tens (neuter):

```
\gdef\@@tenthstringNgerman#1{%
  \ifcase#1%
    \or zehntes%
    \or zwanzigstes%
    \or drei{\ss}igstes%
    \or vierzigstes%
    \or f\"unfzigstes%
    \or sechzigstes%
    \or siebzigstes%
    \or achtzigstes%
    \or neunzigstes%
  \fi
}%
```

Teens (neuter)

```
\gdef\@@teenthstringNgerman#1{%
  \ifcase#1%
    zehntes%
    \or elftes%
    \or zw\"olftes%
    \or dreizehntes%
    \or vierzehntes%
    \or f\"unfzehntes%
    \or sechzehntes%
    \or siebzehntes%
    \or achtzehntes%
    \or neunzehntes%
  \fi
}%
```

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

```
\gdef\@@numberunderhundredgerman#1#2{%
\ifnum#1<10\relax
  \ifnum#1>0\relax
    \eappto#2{\@unitstring{#1}}%
  \fi
\else
  \@tmpstrctr=#1\relax
  \@modulo{\@tmpstrctr}{10}%
  \ifnum#1<20\relax
    \eappto#2{\@teenstring{\@tmpstrctr}}%
  \else
    \ifnum\@tmpstrctr=0\relax
    \else
      \eappto#2{\@unitstring{\@tmpstrctr}und}%
    \fi
    \@tmpstrctr=#1\relax
    \divide\@tmpstrctr by 10\relax
    \eappto#2{\@tenstring{\@tmpstrctr}}%
  \fi
\fi
}%
```

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

```
\gdef\@@numberstringgerman#1#2{%
\ifnum#1>99999\relax
  \PackageError{fmtcount}{Out of range}%
  {This macro only works for values less than 100000}%
\else
  \ifnum#1<0\relax
    \PackageError{fmtcount}{Negative numbers not permitted}%
    {This macro does not work for negative numbers, however
    you can try typing "minus" first, and then pass the modulus of
    this number}%
  \fi
\fi
\def#2{}%
\@strctr=#1\relax \divide\@strctr by 1000\relax
\ifnum\@strctr>1\relax
```

#1 is ≥ 2000, `\@strctr` now contains the number of thousands

```
  \@@numberunderhundredgerman{\@strctr}{#2}%
  \appto#2{tausend}%
\else
```

#1 lies in range [1000,1999]

```
  \ifnum\@strctr=1\relax
    \eappto#2{\eintausend}%
  \fi
\fi
\@strctr=#1\relax
\@modulo{\@strctr}{1000}%
\divide\@strctr by 100\relax
\ifnum\@strctr>1\relax
```

now dealing with number in range [200,999]

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

dealing with number in range [100,199]

```
      \ifnum#1>1000\relax
```

if original number > 1000, use einhundert

```
         \appto#2{einhundert}%
      \else
```

otherwise use `\einhundert`

```
         \eappto#2{\einhundert}%
       \fi
   \fi
\fi
\@strctr=#1\relax
\@modulo{\@strctr}{100}%
\ifnum#1=0\relax
  \def#2{null}%
\else
  \ifnum\@strctr=1\relax
    \appto#2{eins}%
  \else
    \@@numberunderhundredgerman{\@strctr}{#2}%
  \fi
\fi
}%
```

As above, but for ordinals

```
\gdef\@@numberunderhundredthgerman#1#2{%
\ifnum#1<10\relax
 \eappto#2{\@unitthstring{#1}}%
\else
  \@tmpstrctr=#1\relax
  \@modulo{\@tmpstrctr}{10}%
  \ifnum#1<20\relax
    \eappto#2{\@teenthstring{\@tmpstrctr}}%
  \else
    \ifnum\@tmpstrctr=0\relax
    \else
      \eappto#2{\@unitstring{\@tmpstrctr}und}%
    \fi
    \@tmpstrctr=#1\relax
    \divide\@tmpstrctr by 10\relax
    \eappto#2{\@tenthstring{\@tmpstrctr}}%
  \fi
\fi
}%
```

```
\gdef\@@ordinalstringgerman#1#2{%
\ifnum#1>99999\relax
  \PackageError{fmtcount}{Out of range}%
  {This macro only works for values less than 100000}%
\else
  \ifnum#1<0\relax
    \PackageError{fmtcount}{Negative numbers not permitted}%
    {This macro does not work for negative numbers, however
    you can try typing "minus" first, and then pass the modulus of
    this number}%
  \fi
\fi
\def#2{}%
\@strctr=#1\relax \divide\@strctr by 1000\relax
\ifnum\@strctr>1\relax
```

#1 is ≥ 2000, `\@strctr` now contains the number of thousands

```
\@@numberunderhundredgerman{\@strctr}{#2}%
```

is that it, or is there more?

```
  \@tmpstrctr=#1\relax \@modulo{\@tmpstrctr}{1000}%
  \ifnum\@tmpstrctr=0\relax
    \eappto#2{\@thousandth}%
  \else
    \appto#2{tausend}%
  \fi
\else
```

#1 lies in range [1000,1999]

```
  \ifnum\@strctr=1\relax
```

```
    \ifnum#1=1000\relax
      \eappto#2{\@thousandth}%
    \else
      \eappto#2{\eintausend}%
    \fi
  \fi
\fi
\@strctr=#1\relax
\@modulo{\@strctr}{1000}%
\divide\@strctr by 100\relax
\ifnum\@strctr>1\relax
```

now dealing with number in range [200,999] is that it, or is there more?

```
  \@tmpstrctr=#1\relax \@modulo{\@tmpstrctr}{100}%
  \ifnum\@tmpstrctr=0\relax
     \ifnum\@strctr=1\relax
       \eappto#2{\@hundredth}%
     \else
       \eappto#2{\@unitstring{\@strctr}\@hundredth}%
     \fi
  \else
     \eappto#2{\@unitstring{\@strctr}hundert}%
  \fi
\else
   \ifnum\@strctr=1\relax
```

dealing with number in range [100,199] is that it, or is there more?

```
     \@tmpstrctr=#1\relax \@modulo{\@tmpstrctr}{100}%
     \ifnum\@tmpstrctr=0\relax
        \eappto#2{\@hundredth}%
     \else
     \ifnum#1>1000\relax
        \appto#2{einhundert}%
     \else
        \eappto#2{\einhundert}%
     \fi
     \fi
   \fi
\fi
\@strctr=#1\relax
\@modulo{\@strctr}{100}%
\ifthenelse{\@strctr=0 \and #1>0}{}{%
\@@numberunderhundredthgerman{\@strctr}{#2}%
}%
}%
```

Load fc-germanb.def if not already loaded

```
\FCloadlang{germanb}%
```

9.4.8 fc-germanb.def

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

Load fc-german.def if not already loaded

```
\FCloadlang{german}%
```

Set germanb to be equivalent to german.

```
\global\let\@ordinalMgermanb=\@ordinalMgerman
\global\let\@ordinalFgermanb=\@ordinalFgerman
\global\let\@ordinalNgermanb=\@ordinalNgerman
\global\let\@numberstringMgermanb=\@numberstringMgerman
\global\let\@numberstringFgermanb=\@numberstringFgerman
\global\let\@numberstringNgermanb=\@numberstringNgerman
\global\let\@NumberstringMgermanb=\@NumberstringMgerman
\global\let\@NumberstringFgermanb=\@NumberstringFgerman
\global\let\@NumberstringNgermanb=\@NumberstringNgerman
\global\let\@ordinalstringMgermanb=\@ordinalstringMgerman
\global\let\@ordinalstringFgermanb=\@ordinalstringFgerman
\global\let\@ordinalstringNgermanb=\@ordinalstringNgerman
\global\let\@OrdinalstringMgermanb=\@OrdinalstringMgerman
\global\let\@OrdinalstringFgermanb=\@OrdinalstringFgerman
\global\let\@OrdinalstringNgermanb=\@OrdinalstringNgerman
```

9.4.9 fc-italian

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

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

\RequirePackage{itnumpar}

\newcommand{\@numberstringMitalian}[2]{%
  \edef#2{\noexpand\printnumeroinparole{#1}}%
}
\global\let\@numberstringMitalian\@numberstringMitalian

\newcommand{\@numberstringFitalian}[2]{%
  \edef#2{\noexpand\printnumeroinparole{#1}}}

\global\let\@numberstringFitalian\@numberstringFitalian

\newcommand{\@NumberstringMitalian}[2]{%
  \edef#2{\noexpand\printNumeroinparole{#1}}%
}
\global\let\@NumberstringMitalian\@NumberstringMitalian

\newcommand{\@NumberstringFitalian}[2]{%
  \edef#2{\noexpand\printNumeroinparole{#1}}%
}
\global\let\@NumberstringFitalian\@NumberstringFitalian

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

```
  \edef#2{\noexpand\printordinalem{#1}}%
}
\global\let\@ordinalstringMitalian\@ordinalstringMitalian

\newcommand{\@ordinalstringFitalian}[2]{%
  \edef#2{\noexpand\printordinalef{#1}}%
}
\global\let\@ordinalstringFitalian\@ordinalstringFitalian

\newcommand{\@OrdinalstringMitalian}[2]{%
  \edef#2{\noexpand\printOrdinalem{#1}}%
}
\global\let\@OrdinalstringMitalian\@OrdinalstringMitalian

\newcommand{\@OrdinalstringFitalian}[2]{%
  \edef#2{\noexpand\printOrdinalef{#1}}%
}
\global\let\@OrdinalstringFitalian\@OrdinalstringFitalian

\newcommand{\@ordinalMitalian}[2]{%
  \edef#2{#1\relax\noexpand\fmtord{o}}}

\global\let\@ordinalMitalian\@ordinalMitalian

\newcommand{\@ordinalFitalian}[2]{%
  \edef#2{#1\relax\noexpand\fmtord{a}}}
\global\let\@ordinalFitalian\@ordinalFitalian
```

9.4.10 fc-ngerman.def

```
\ProvidesFCLanguage{ngerman}[2012/06/18]%
\FCloadlang{german}%
\FCloadlang{ngermanb}%
```

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

```
\global\let\@ordinalMngerman=\@ordinalMgerman
\global\let\@ordinalFngerman=\@ordinalFgerman
\global\let\@ordinalNngerman=\@ordinalNgerman
\global\let\@numberstringMngerman=\@numberstringMgerman
\global\let\@numberstringFngerman=\@numberstringFgerman
\global\let\@numberstringNngerman=\@numberstringNgerman
\global\let\@NumberstringMngerman=\@NumberstringMgerman
\global\let\@NumberstringFngerman=\@NumberstringFgerman
\global\let\@NumberstringNngerman=\@NumberstringNgerman
\global\let\@ordinalstringMngerman=\@ordinalstringMgerman
\global\let\@ordinalstringFngerman=\@ordinalstringFgerman
\global\let\@ordinalstringNngerman=\@ordinalstringNgerman
\global\let\@OrdinalstringMngerman=\@OrdinalstringMgerman
\global\let\@OrdinalstringFngerman=\@OrdinalstringFgerman
```

```
\global\let\@OrdinalstringNngerman=\@OrdinalstringNgerman
```

9.4.11 fc-ngermanb.def

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

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

```
\global\let\@ordinalMngermanb=\@ordinalMgerman
\global\let\@ordinalFngermanb=\@ordinalFgerman
\global\let\@ordinalNngermanb=\@ordinalNgerman
\global\let\@numberstringMngermanb=\@numberstringMgerman
\global\let\@numberstringFngermanb=\@numberstringFgerman
\global\let\@numberstringNngermanb=\@numberstringNgerman
\global\let\@NumberstringMngermanb=\@NumberstringMgerman
\global\let\@NumberstringFngermanb=\@NumberstringFgerman
\global\let\@NumberstringNngermanb=\@NumberstringNgerman
\global\let\@ordinalstringMngermanb=\@ordinalstringMgerman
\global\let\@ordinalstringFngermanb=\@ordinalstringFgerman
\global\let\@ordinalstringNngermanb=\@ordinalstringNgerman
\global\let\@OrdinalstringMngermanb=\@OrdinalstringMgerman
\global\let\@OrdinalstringFngermanb=\@OrdinalstringFgerman
\global\let\@OrdinalstringNngermanb=\@OrdinalstringNgerman
```

Load fc-ngerman.def if not already loaded

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

9.4.12 fc-portuges.def

Portuguse definitions

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

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

```
\gdef\@ordinalMportuges#1#2{%
  \ifnum#1=0\relax
    \edef#2{\number#1}%
  \else
    \edef#2{\number#1\relax\noexpand\fmtord{o}}%
  \fi
}%
```

Feminine:

```
\gdef\@ordinalFportuges#1#2{%
  \ifnum#1=0\relax
    \edef#2{\number#1}%
  \else
    \edef#2{\number#1\relax\noexpand\fmtord{a}}%
  \fi
}%
```

Make neuter same as masculine:

```
\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):

```
\gdef\@@unitstringportuges#1#2{%
  \ifcase#1\relax
    zero%
    \or um%
    \or dois%
    \or tr\^es%
    \or quatro%
    \or cinco%
    \or seis%
    \or sete%
    \or oito%
    \or nove%
  \fi
}%
%   \end{macrocode}
% As above, but for feminine:
%   \begin{macrocode}
\gdef\@@unitstringFportuges#1{%
  \ifcase#1\relax
    zero%
    \or uma%
    \or duas%
    \or tr\^es%
    \or quatro%
    \or cinco%
    \or seis%
    \or sete%
    \or oito%
    \or nove%
  \fi
}%
```

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

```
\gdef\@@tenstringportuges#1{%
  \ifcase#1\relax
    \or dez%
    \or vinte%
    \or trinta%
    \or quarenta%
    \or cinq\"uenta%
    \or sessenta%
    \or setenta%
    \or oitenta%
    \or noventa%
    \or cem%
  \fi
}%
```

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

```
\gdef\@@teenstringportuges#1{%
  \ifcase#1\relax
    dez%
    \or onze%
    \or doze%
    \or treze%
    \or quatorze%
    \or quinze%
    \or dezesseis%
    \or dezessete%
    \or dezoito%
    \or dezenove%
  \fi
}%
```

Hundreds:

```
\gdef\@@hundredstringportuges#1{%
  \ifcase#1\relax
    \or cento%
    \or duzentos%
    \or trezentos%
    \or quatrocentos%
    \or quinhentos%
    \or seiscentos%
    \or setecentos%
    \or oitocentos%
    \or novecentos%
  \fi
}%
```

Hundreds (feminine):

```
\gdef\@@hundredstringFportuges#1{%
  \ifcase#1\relax
    \or cento%
    \or duzentas%
    \or trezentas%
    \or quatrocentas%
    \or quinhentas%
    \or seiscentas%
    \or setecentas%
    \or oitocentas%
    \or novecentas%
  \fi
}%
```

Units (initial letter in upper case):

```
\gdef\@@Unitstringportuges#1{%
  \ifcase#1\relax
    Zero%
    \or Um%
    \or Dois%
    \or Tr\^es%
    \or Quatro%
    \or Cinco%
    \or Seis%
    \or Sete%
    \or Oito%
    \or Nove%
  \fi
}%
```

As above, but feminine:

```
\gdef\@@UnitstringFportuges#1{%
  \ifcase#1\relax
    Zera%
    \or Uma%
    \or Duas%
    \or Tr\^es%
    \or Quatro%
    \or Cinco%
    \or Seis%
    \or Sete%
    \or Oito%
    \or Nove%
  \fi
}%
```

Tens (with initial letter in upper case):

```
\gdef\@@Tenstringportuges#1{%
  \ifcase#1\relax
    \or Dez%
    \or Vinte%
    \or Trinta%
    \or Quarenta%
    \or Cinq\"uenta%
    \or Sessenta%
    \or Setenta%
    \or Oitenta%
    \or Noventa%
    \or Cem%
  \fi
}%
```

Teens (with initial letter in upper case):

```
\gdef\@@Teenstringportuges#1{%
  \ifcase#1\relax
    Dez%
    \or Onze%
    \or Doze%
    \or Treze%
    \or Quatorze%
    \or Quinze%
    \or Dezesseis%
    \or Dezessete%
    \or Dezoito%
    \or Dezenove%
  \fi
}%
```

Hundreds (with initial letter in upper case):

```
\gdef\@@Hundredstringportuges#1{%
  \ifcase#1\relax
    \or Cento%
    \or Duzentos%
    \or Trezentos%
    \or Quatrocentos%
    \or Quinhentos%
    \or Seiscentos%
    \or Setecentos%
    \or Oitocentos%
    \or Novecentos%
  \fi
}%
```

As above, but feminine:

```
\gdef\@@HundredstringFportuges#1{%
  \ifcase#1\relax
    \or Cento%
    \or Duzentas%
    \or Trezentas%
    \or Quatrocentas%
    \or Quinhentas%
    \or Seiscentas%
    \or Setecentas%
    \or Oitocentas%
    \or Novecentas%
  \fi
}%
```

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

```
\DeclareRobustCommand{\@numberstringMportuges}[2]{%
  \let\@unitstring=\@@unitstringportuges
  \let\@teenstring=\@@teenstringportuges
  \let\@tenstring=\@@tenstringportuges
  \let\@hundredstring=\@@hundredstringportuges
  \def\@hundred{cem}\def\@thousand{mil}%
  \def\@andname{e}%
  \@@numberstringportuges{#1}{#2}%
}%
\global\let\@numberstringMportuges\@numberstringMportuges
```

As above, but feminine form:

```
\DeclareRobustCommand{\@numberstringFportuges}[2]{%
  \let\@unitstring=\@@unitstringFportuges
  \let\@teenstring=\@@teenstringportuges
  \let\@tenstring=\@@tenstringportuges
  \let\@hundredstring=\@@hundredstringFportuges
  \def\@hundred{cem}\def\@thousand{mil}%
  \def\@andname{e}%
  \@@numberstringportuges{#1}{#2}%
}%
\global\let\@numberstringFportuges\@numberstringFportuges
```

Make neuter same as masculine:

```
\global\let\@numberstringNportuges\@numberstringMportuges
```

As above, but initial letters in upper case:

```
\DeclareRobustCommand{\@NumberstringMportuges}[2]{%
  \let\@unitstring=\@@Unitstringportuges
  \let\@teenstring=\@@Teenstringportuges
  \let\@tenstring=\@@Tenstringportuges
  \let\@hundredstring=\@@Hundredstringportuges
  \def\@hundred{Cem}\def\@thousand{Mil}%
  \def\@andname{e}%
  \@@numberstringportuges{#1}{#2}%
}%
\global\let\@NumberstringMportuges\@NumberstringMportuges
```

As above, but feminine form:

```
\DeclareRobustCommand{\@NumberstringFportuges}[2]{%
  \let\@unitstring=\@@UnitstringFportuges
  \let\@teenstring=\@@Teenstringportuges
  \let\@tenstring=\@@Tenstringportuges
  \let\@hundredstring=\@@HundredstringFportuges
  \def\@hundred{Cem}\def\@thousand{Mil}%
  \def\@andname{e}%
  \@@numberstringportuges{#1}{#2}%
}%
\global\let\@NumberstringFportuges\@NumberstringFportuges
```

Make neuter same as masculine:

```
\global\let\@NumberstringNportuges\@NumberstringMportuges
```

As above, but for ordinals.

```
\DeclareRobustCommand{\@ordinalstringMportuges}[2]{%
  \let\@unitthstring=\@@unitthstringportuges
  \let\@unitstring=\@@unitstringportuges
  \let\@teenthstring=\@@teenthstringportuges
  \let\@tenthstring=\@@tenthstringportuges
  \let\@hundredthstring=\@@hundredthstringportuges
  \def\@thousandth{mil\'esimo}%
  \@@ordinalstringportuges{#1}{#2}%
}%
\global\let\@ordinalstringMportuges\@ordinalstringMportuges
```

Feminine form:

```
\DeclareRobustCommand{\@ordinalstringFportuges}[2]{%
  \let\@unitthstring=\@@unitthstringFportuges
  \let\@unitstring=\@@unitstringFportuges
  \let\@teenthstring=\@@teenthstringportuges
  \let\@tenthstring=\@@tenthstringFportuges
  \let\@hundredthstring=\@@hundredthstringFportuges
  \def\@thousandth{mil\'esima}%
  \@@ordinalstringportuges{#1}{#2}%
}%
\global\let\@ordinalstringFportuges\@ordinalstringFportuges
```

Make neuter same as masculine:

```
\global\let\@ordinalstringNportuges\@ordinalstringMportuges
```

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

```
\DeclareRobustCommand{\@OrdinalstringMportuges}[2]{%
  \let\@unitthstring=\@@Unitthstringportuges
  \let\@unitstring=\@@Unitstringportuges
  \let\@teenthstring=\@@teenthstringportuges
  \let\@tenthstring=\@@Tenthstringportuges
  \let\@hundredthstring=\@@Hundredthstringportuges
  \def\@thousandth{Mil\'esimo}%
  \@@ordinalstringportuges{#1}{#2}%
}%
\global\let\@OrdinalstringMportuges\@OrdinalstringMportuges
```

Feminine form:

```
\DeclareRobustCommand{\@OrdinalstringFportuges}[2]{%
  \let\@unitthstring=\@@UnitthstringFportuges
  \let\@unitstring=\@@UnitstringFportuges
  \let\@teenthstring=\@@teenthstringportuges
  \let\@tenthstring=\@@TenthstringFportuges
  \let\@hundredthstring=\@@HundredthstringFportuges
  \def\@thousandth{Mil\'esima}%
  \@@ordinalstringportuges{#1}{#2}%
}%
\global\let\@OrdinalstringFportuges\@OrdinalstringFportuges
```

Make neuter same as masculine:

```
\global\let\@OrdinalstringNportuges\@OrdinalstringMportuges
```

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

```
\gdef\@@unitthstringportuges#1{%
  \ifcase#1\relax
    zero%
    \or primeiro%
    \or segundo%
    \or terceiro%
    \or quarto%
    \or quinto%
    \or sexto%
    \or s\'etimo%
    \or oitavo%
    \or nono%
  \fi
}%
```

Tens:

```
\gdef\@@tenthstringportuges#1{%
  \ifcase#1\relax
    \or d\'ecimo%
    \or vig\'esimo%
    \or trig\'esimo%
    \or quadrag\'esimo%
    \or q\"uinquag\'esimo%
    \or sexag\'esimo%
    \or setuag\'esimo%
    \or octog\'esimo%
    \or nonag\'esimo%
  \fi
}%
```

Teens:

```
\gdef\@@teenthstringportuges#1{%
  \@tenthstring{1}%
  \ifnum#1>0\relax
    -\@unitthstring{#1}%
  \fi
}%
```

Hundreds:

```
\gdef\@@hundredthstringportuges#1{%
  \ifcase#1\relax
    \or cent\'esimo%
    \or ducent\'esimo%
    \or trecent\'esimo%
    \or quadringent\'esimo%
    \or q\"uingent\'esimo%
    \or seiscent\'esimo%
    \or setingent\'esimo%
    \or octingent\'esimo%
    \or nongent\'esimo%
  \fi
}%
```

Units (feminine):

```
\gdef\@@unitthstringFportuges#1{%
  \ifcase#1\relax
    zero%
    \or primeira%
    \or segunda%
    \or terceira%
    \or quarta%
    \or quinta%
    \or sexta%
    \or s\'etima%
    \or oitava%
    \or nona%
  \fi
}%
```

Tens (feminine):

```
\gdef\@@tenthstringFportuges#1{%
  \ifcase#1\relax
    \or d\'ecima%
    \or vig\'esima%
    \or trig\'esima%
    \or quadrag\'esima%
    \or q\"uinquag\'esima%
    \or sexag\'esima%
    \or setuag\'esima%
    \or octog\'esima%
    \or nonag\'esima%
  \fi
}%
```

Hundreds (feminine):

```
\gdef\@@hundredthstringFportuges#1{%
  \ifcase#1\relax
    \or cent\'esima%
    \or ducent\'esima%
    \or trecent\'esima%
    \or quadringent\'esima%
    \or q\"uingent\'esima%
    \or seiscent\'esima%
    \or setingent\'esima%
    \or octingent\'esima%
    \or nongent\'esima%
  \fi
}%
```

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

```
\gdef\@@Unitthstringportuges#1{%
  \ifcase#1\relax
```

```
    Zero%
    \or Primeiro%
    \or Segundo%
    \or Terceiro%
    \or Quarto%
    \or Quinto%
    \or Sexto%
    \or S\'etimo%
    \or Oitavo%
    \or Nono%
  \fi
}%
```

Tens:

```
\gdef\@@Tenthstringportuges#1{%
  \ifcase#1\relax
    \or D\'ecimo%
    \or Vig\'esimo%
    \or Trig\'esimo%
    \or Quadrag\'esimo%
    \or Q\"uinquag\'esimo%
    \or Sexag\'esimo%
    \or Setuag\'esimo%
    \or Octog\'esimo%
    \or Nonag\'esimo%
  \fi
}%
```

Hundreds:

```
\gdef\@@Hundredthstringportuges#1{%
  \ifcase#1\relax
    \or Cent\'esimo%
    \or Ducent\'esimo%
    \or Trecent\'esimo%
    \or Quadringent\'esimo%
    \or Q\"uingent\'esimo%
    \or Seiscent\'esimo%
    \or Setingent\'esimo%
    \or Octingent\'esimo%
    \or Nongent\'esimo%
  \fi
}%
```

As above, but feminine. Units:

```
\gdef\@@UnitthstringFportuges#1{%
  \ifcase#1\relax
    Zera%
    \or Primeira%
    \or Segunda%
    \or Terceira%
    \or Quarta%
    \or Quinta%
    \or Sexta%
    \or S\'etima%
    \or Oitava%
    \or Nona%
  \fi
}%
```

Tens (feminine);

```
\gdef\@@TenthstringFportuges#1{%
  \ifcase#1\relax
    \or D\'ecima%
    \or Vig\'esima%
    \or Trig\'esima%
    \or Quadrag\'esima%
    \or Q\"uinquag\'esima%
    \or Sexag\'esima%
    \or Setuag\'esima%
    \or Octog\'esima%
    \or Nonag\'esima%
  \fi
}%
```

Hundreds (feminine):

```
\gdef\@@HundredthstringFportuges#1{%
  \ifcase#1\relax
    \or Cent\'esima%
    \or Ducent\'esima%
    \or Trecent\'esima%
    \or Quadringent\'esima%
    \or Q\"uingent\'esima%
    \or Seiscent\'esima%
    \or Setingent\'esima%
    \or Octingent\'esima%
    \or Nongent\'esima%
  \fi
}%
```

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

```
\gdef\@@numberstringportuges#1#2{%
\ifnum#1>99999
\PackageError{fmtcount}{Out of range}%
{This macro only works for values less than 100000}%
\else
\ifnum#1<0
\PackageError{fmtcount}{Negative numbers not permitted}%
{This macro does not work for negative numbers, however
you can try typing "minus" first, and then pass the modulus of
this number}%
\fi
\fi
\def#2{}%
\@strctr=#1\relax \divide\@strctr by 1000\relax
\ifnum\@strctr>9
```

#1 is greater or equal to 10000

```
  \divide\@strctr by 10
  \ifnum\@strctr>1\relax
    \let\@@fc@numstr#2\relax
    \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
    \@strctr=#1 \divide\@strctr by 1000\relax
    \@modulo{\@strctr}{10}%
    \ifnum\@strctr>0
      \ifnum\@strctr=1\relax
        \let\@@fc@numstr#2\relax
        \edef#2{\@@fc@numstr\ \@andname}%
      \fi
      \let\@@fc@numstr#2\relax
      \edef#2{\@@fc@numstr\ \@unitstring{\@strctr}}%
    \fi
  \else
    \@strctr=#1\relax
    \divide\@strctr by 1000\relax
    \@modulo{\@strctr}{10}%
    \let\@@fc@numstr#2\relax
    \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
  \fi
  \let\@@fc@numstr#2\relax
  \edef#2{\@@fc@numstr\ \@thousand}%
\else
  \ifnum\@strctr>0\relax
    \ifnum\@strctr>1\relax
      \let\@@fc@numstr#2\relax
      \edef#2{\@@fc@numstr\@unitstring{\@strctr}\ }%
    \fi
    \let\@@fc@numstr#2\relax
    \edef#2{\@@fc@numstr\@thousand}%
  \fi
\fi
\@strctr=#1\relax \@modulo{\@strctr}{1000}%
\divide\@strctr by 100\relax
\ifnum\@strctr>0\relax
  \ifnum#1>1000 \relax
    \let\@@fc@numstr#2\relax
    \edef#2{\@@fc@numstr\ }%
  \fi
  \@tmpstrctr=#1\relax
  \@modulo{\@tmpstrctr}{1000}%
  \let\@@fc@numstr#2\relax
  \ifnum\@tmpstrctr=100\relax
    \edef#2{\@@fc@numstr\@tenstring{10}}%
  \else
    \edef#2{\@@fc@numstr\@hundredstring{\@strctr}}%
  \fi%
\fi
\@strctr=#1\relax \@modulo{\@strctr}{100}%
\ifnum#1>100\relax
  \ifnum\@strctr>0\relax
    \let\@@fc@numstr#2\relax
    \edef#2{\@@fc@numstr\ \@andname\ }%
  \fi
\fi
\ifnum\@strctr>19\relax
  \divide\@strctr by 10\relax
  \let\@@fc@numstr#2\relax
  \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
  \@strctr=#1\relax \@modulo{\@strctr}{10}%
  \ifnum\@strctr>0
    \ifnum\@strctr=1\relax
      \let\@@fc@numstr#2\relax
      \edef#2{\@@fc@numstr\ \@andname}%
    \else
      \ifnum#1>100\relax
        \let\@@fc@numstr#2\relax
        \edef#2{\@@fc@numstr\ \@andname}%
      \fi
    \fi
    \let\@@fc@numstr#2\relax
    \edef#2{\@@fc@numstr\ \@unitstring{\@strctr}}%
  \fi
\else
  \ifnum\@strctr<10\relax
    \ifnum\@strctr=0\relax
      \ifnum#1<100\relax
        \let\@@fc@numstr#2\relax
        \edef#2{\@@fc@numstr\@unitstring{\@strctr}}%
      \fi
    \else%(>0,<10)
      \let\@@fc@numstr#2\relax
      \edef#2{\@@fc@numstr\@unitstring{\@strctr}}%
    \fi
  \else%>10
    \@modulo{\@strctr}{10}%
    \let\@@fc@numstr#2\relax
    \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
  \fi
```

```
\fi
}%
```

As above, but for ordinals.

```
\gdef\@@ordinalstringportuges#1#2{%
\@strctr=#1\relax
\ifnum#1>99999
\PackageError{fmtcount}{Out of range}%
{This macro only works for values less than 100000}%
\else
\ifnum#1<0
\PackageError{fmtcount}{Negative numbers not permitted}%
{This macro does not work for negative numbers, however
you can try typing "minus" first, and then pass the modulus of
this number}%
\else
\def#2{}%
\ifnum\@strctr>999\relax
  \divide\@strctr by 1000\relax
  \ifnum\@strctr>1\relax
    \ifnum\@strctr>9\relax
      \@tmpstrctr=\@strctr
      \ifnum\@strctr<20
        \@modulo{\@tmpstrctr}{10}%
        \let\@@fc@ordstr#2\relax
        \edef#2{\@@fc@ordstr\@teenthstring{\@tmpstrctr}}%
      \else
        \divide\@tmpstrctr by 10\relax
        \let\@@fc@ordstr#2\relax
        \edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
        \@tmpstrctr=\@strctr
        \@modulo{\@tmpstrctr}{10}%
        \ifnum\@tmpstrctr>0\relax
          \let\@@fc@ordstr#2\relax
          \edef#2{\@@fc@ordstr\@unitthstring{\@tmpstrctr}}%
        \fi
      \fi
    \else
      \let\@@fc@ordstr#2\relax
      \edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
    \fi
  \fi
  \let\@@fc@ordstr#2\relax
  \edef#2{\@@fc@ordstr\@thousandth}%
\fi
\@strctr=#1\relax
\@modulo{\@strctr}{1000}%
\ifnum\@strctr>99\relax
  \@tmpstrctr=\@strctr
  \divide\@tmpstrctr by 100\relax
  \ifnum#1>1000\relax
    \let\@@fc@ordstr#2\relax
    \edef#2{\@@fc@ordstr-}%
  \fi
  \let\@@fc@ordstr#2\relax
  \edef#2{\@@fc@ordstr\@hundredthstring{\@tmpstrctr}}%
\fi
\@modulo{\@strctr}{100}%
\ifnum#1>99\relax
  \ifnum\@strctr>0\relax
    \let\@@fc@ordstr#2\relax
    \edef#2{\@@fc@ordstr-}%
  \fi
\fi
\ifnum\@strctr>9\relax
  \@tmpstrctr=\@strctr
  \divide\@tmpstrctr by 10\relax
  \let\@@fc@ordstr#2\relax
  \edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
  \@tmpstrctr=\@strctr
  \@modulo{\@tmpstrctr}{10}%
  \ifnum\@tmpstrctr>0\relax
    \let\@@fc@ordstr#2\relax
    \edef#2{\@@fc@ordstr-\@unitthstring{\@tmpstrctr}}%
  \fi
\else
  \ifnum\@strctr=0\relax
    \ifnum#1=0\relax
      \let\@@fc@ordstr#2\relax
      \edef#2{\@@fc@ordstr\@unitstring{0}}%
    \fi
  \else
    \let\@@fc@ordstr#2\relax
    \edef#2{\@@fc@ordstr\@unitthstring{\@strctr}}%
  \fi
\fi
\fi
\fi
}%
```

9.4.13 fc-spanish.def

Spanish definitions

```
\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:

```
\gdef\@ordinalMspanish#1{%
```

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

Feminine:

```
\gdef\@ordinalFspanish}[2]{%
  \edef#2{\number#1\relax\noexpand\fmtord{a}}%
}%
```

Make neuter same as masculine:

```
\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):

```
\gdef\@@unitstringspanish#1{%
  \ifcase#1\relax
    cero%
    \or uno%
    \or dos%
    \or tres%
    \or cuatro%
    \or cinco%
    \or seis%
    \or siete%
    \or ocho%
    \or nueve%
  \fi
}%
```

Feminine:

```
\gdef\@@unitstringFspanish#1{%
  \ifcase#1\relax
    cera%
    \or una%
    \or dos%
    \or tres%
    \or cuatro%
    \or cinco%
    \or seis%
    \or siete%
    \or ocho%
    \or nueve%
  \fi
}%
```

Tens (argument must go from 1 to 10):

```
\gdef\@@tenstringspanish#1{%
  \ifcase#1\relax
    \or diez%
    \or veinte%
    \or treinta%
    \or cuarenta%
    \or cincuenta%
    \or sesenta%
    \or setenta%
    \or ochenta%
    \or noventa%
    \or cien%
  \fi
}%
```

Teens:

```
\gdef\@@teenstringspanish#1{%
  \ifcase#1\relax
    diez%
    \or once%
    \or doce%
    \or trece%
    \or catorce%
    \or quince%
    \or diecis\'eis%
    \or diecisiete%
    \or dieciocho%
    \or diecinueve%
  \fi
}%
```

Twenties:

```
\gdef\@@twentystringspanish#1{%
  \ifcase#1\relax
    veinte%
    \or veintiuno%
    \or veintid\'os%
    \or veintitr\'es%
    \or veinticuatro%
    \or veinticinco%
    \or veintis\'eis%
    \or veintisiete%
    \or veintiocho%
    \or veintinueve%
  \fi
}%
```

Feminine form:

```
\gdef\@@twentystringFspanish#1{%
  \ifcase#1\relax
    veinte%
    \or veintiuna%
    \or veintid\'os%
    \or veintitr\'es%
    \or veinticuatro%
    \or veinticinco%
    \or veintis\'eis%
    \or veintisiete%
    \or veintiocho%
    \or veintinueve%
  \fi
}%
```

Hundreds:

```
\gdef\@@hundredstringspanish#1{%
  \ifcase#1\relax
    \or ciento%
    \or doscientos%
    \or trescientos%
    \or cuatrocientos%
    \or quinientos%
    \or seiscientos%
    \or setecientos%
    \or ochocientos%
    \or novecientos%
  \fi
}%
```

Feminine form:

```
\gdef\@@hundredstringFspanish#1{%
  \ifcase#1\relax
    \or cienta%
    \or doscientas%
    \or trescientas%
    \or cuatrocientas%
    \or quinientas%
    \or seiscientas%
    \or setecientas%
    \or ochocientas%
    \or novecientas%
  \fi
}%
```

As above, but with initial letter uppercase:

```
\gdef\@@Unitstringspanish#1{%
  \ifcase#1\relax
    Cero%
    \or Uno%
    \or Dos%
    \or Tres%
    \or Cuatro%
    \or Cinco%
    \or Seis%
    \or Siete%
    \or Ocho%
    \or Nueve%
  \fi
}%
```

Feminine form:

```
\gdef\@@UnitstringFspanish#1{%
  \ifcase#1\relax
    Cera%
    \or Una%
    \or Dos%
    \or Tres%
    \or Cuatro%
    \or Cinco%
    \or Seis%
    \or Siete%
    \or Ocho%
    \or Nueve%
  \fi
}%
```

Tens:

```
%\changes{2.0}{2012-06-18}{fixed spelling mistake (correction
%provided by Fernando Maldonado)}
\gdef\@@Tenstringspanish#1{%
  \ifcase#1\relax
    \or Diez%
    \or Veinte%
    \or Treinta%
    \or Cuarenta%
    \or Cincuenta%
    \or Sesenta%
    \or Setenta%
    \or Ochenta%
    \or Noventa%
    \or Cien%
  \fi
}%
```

Teens:

```
\gdef\@@Teenstringspanish#1{%
  \ifcase#1\relax
    Diez%
    \or Once%
    \or Doce%
    \or Trece%
    \or Catorce%
    \or Quince%
    \or Diecis\'eis%
    \or Diecisiete%
    \or Dieciocho%
    \or Diecinueve%
  \fi
}%
```

Twenties:

```
\gdef\@@Twentystringspanish#1{%
  \ifcase#1\relax
    Veinte%
    \or Veintiuno%
    \or Veintid\'os%
    \or Veintitr\'es%
    \or Veinticuatro%
    \or Veinticinco%
    \or Veintis\'eis%
    \or Veintisiete%
    \or Veintiocho%
    \or Veintinueve%
  \fi
}%
```

Feminine form:

```
\gdef\@@TwentystringFspanish#1{%
  \ifcase#1\relax
    Veinte%
    \or Veintiuna%
    \or Veintid\'os%
    \or Veintitr\'es%
    \or Veinticuatro%
    \or Veinticinco%
    \or Veintis\'eis%
    \or Veintisiete%
    \or Veintiocho%
    \or Veintinueve%
  \fi
}%
```

Hundreds:

```
\gdef\@@Hundredstringspanish#1{%
  \ifcase#1\relax
    \or Ciento%
    \or Doscientos%
    \or Trescientos%
    \or Cuatrocientos%
    \or Quinientos%
    \or Seiscientos%
    \or Setecientos%
    \or Ochocientos%
    \or Novecientos%
  \fi
}%
```

Feminine form:

```
\gdef\@@HundredstringFspanish#1{%
  \ifcase#1\relax
```

```
    \or Cienta%
    \or Doscientas%
    \or Trescientas%
    \or Cuatrocientas%
    \or Quinientas%
    \or Seiscientas%
    \or Setecientas%
    \or Ochocientas%
    \or Novecientas%
  \fi
}%
```

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

```
\DeclareRobustCommand{\@numberstringMspanish}[2]{%
  \let\@unitstring=\@@unitstringspanish
  \let\@teenstring=\@@teenstringspanish
  \let\@tenstring=\@@tenstringspanish
  \let\@twentystring=\@@twentystringspanish
  \let\@hundredstring=\@@hundredstringspanish
  \def\@hundred{cien}\def\@thousand{mil}%
  \def\@andname{y}%
  \@@numberstringspanish{#1}{#2}%
}%
\global\let\@numberstringMspanish\@numberstringMspanish
```

Feminine form:

```
\DeclareRobustCommand{\@numberstringFspanish}[2]{%
  \let\@unitstring=\@@unitstringFspanish
  \let\@teenstring=\@@teenstringspanish
  \let\@tenstring=\@@tenstringspanish
  \let\@twentystring=\@@twentystringFspanish
  \let\@hundredstring=\@@hundredstringFspanish
  \def\@hundred{cien}\def\@thousand{mil}%
  \def\@andname{b}%
  \@@numberstringspanish{#1}{#2}%
}%
\global\let\@numberstringFspanish\@numberstringFspanish
```

Make neuter same as masculine:

```
\global\let\@numberstringNspanish\@numberstringMspanish
```

As above, but initial letters in upper case:

```
\DeclareRobustCommand{\@NumberstringMspanish}[2]{%
  \let\@unitstring=\@@Unitstringspanish
  \let\@teenstring=\@@Teenstringspanish
  \let\@tenstring=\@@Tenstringspanish
  \let\@twentystring=\@@Twentystringspanish
  \let\@hundredstring=\@@Hundredstringspanish
  \def\@andname{y}%
  \def\@hundred{Cien}\def\@thousand{Mil}%
  \@@numberstringspanish{#1}{#2}%
}%
\global\let\@NumberstringMspanish\@NumberstringMspanish
```

Feminine form:

```
\DeclareRobustCommand{\@NumberstringFspanish}[2]{%
  \let\@unitstring=\@@UnitstringFspanish
  \let\@teenstring=\@@Teenstringspanish
  \let\@tenstring=\@@Tenstringspanish
  \let\@twentystring=\@@TwentystringFspanish
  \let\@hundredstring=\@@HundredstringFspanish
  \def\@andname{b}%
  \def\@hundred{Cien}\def\@thousand{Mil}%
  \@@numberstringspanish{#1}{#2}%
}%
\global\let\@NumberstringFspanish\@NumberstringFspanish
```

Make neuter same as masculine:

```
\global\let\@NumberstringNspanish\@NumberstringMspanish
```

As above, but for ordinals.

```
\DeclareRobustCommand{\@ordinalstringMspanish}[2]{%
  \let\@unitthstring=\@@unitthstringspanish
  \let\@unitstring=\@@unitstringspanish
  \let\@teenthstring=\@@teenthstringspanish
  \let\@tenthstring=\@@tenthstringspanish
  \let\@hundredthstring=\@@hundredthstringspanish
  \def\@thousandth{mil\'esimo}%
  \@@ordinalstringspanish{#1}{#2}%
}%
\global\let\@ordinalstringMspanish\@ordinalstringMspanish
```

Feminine form:

```
\DeclareRobustCommand{\@ordinalstringFspanish}[2]{%
  \let\@unitthstring=\@@unitthstringFspanish
  \let\@unitstring=\@@unitstringFspanish
  \let\@teenthstring=\@@teenthstringFspanish
  \let\@tenthstring=\@@tenthstringFspanish
  \let\@hundredthstring=\@@hundredthstringFspanish
  \def\@thousandth{mil\'esima}%
  \@@ordinalstringspanish{#1}{#2}%
}%
\global\let\@ordinalstringFspanish\@ordinalstringFspanish
```

Make neuter same as masculine:

```
\global\let\@ordinalstringNspanish\@ordinalstringMspanish
```

As above, but with initial letters in upper case.

```
\DeclareRobustCommand{\@OrdinalstringMspanish}[2]{%
  \let\@unitthstring=\@@Unitthstringspanish
  \let\@unitstring=\@@Unitstringspanish
  \let\@teenthstring=\@@Teenthstringspanish
  \let\@tenthstring=\@@Tenthstringspanish
  \let\@hundredthstring=\@@Hundredthstringspanish
  \def\@thousandth{Mil\'esimo}%
  \@@ordinalstringspanish{#1}{#2}%
}
\global\let\@OrdinalstringMspanish\@OrdinalstringMspanish
```

Feminine form:

```
\DeclareRobustCommand{\@OrdinalstringFspanish}[2]{%
  \let\@unitthstring=\@@UnitthstringFspanish
  \let\@unitstring=\@@UnitstringFspanish
  \let\@teenthstring=\@@TeenthstringFspanish
  \let\@tenthstring=\@@TenthstringFspanish
  \let\@hundredthstring=\@@HundredthstringFspanish
  \def\@thousandth{Mil\'esima}%
  \@@ordinalstringspanish{#1}{#2}%
}%
\global\let\@OrdinalstringFspanish\@OrdinalstringFspanish
```

Make neuter same as masculine:

```
\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:

```
\gdef\@@unitthstringspanish#1{%
  \ifcase#1\relax
    cero%
    \or primero%
    \or segundo%
    \or tercero%
    \or cuarto%
    \or quinto%
    \or sexto%
    \or s\'eptimo%
    \or octavo%
    \or noveno%
  \fi
}%
```

Tens:

```
\gdef\@@tenthstringspanish#1{%
  \ifcase#1\relax
    \or d\'ecimo%
    \or vig\'esimo%
    \or trig\'esimo%
    \or cuadrag\'esimo%
    \or quincuag\'esimo%
    \or sexag\'esimo%
    \or septuag\'esimo%
    \or octog\'esimo%
    \or nonag\'esimo%
  \fi
}%
```

Teens:

```
\gdef\@@teenthstringspanish#1{%
  \ifcase#1\relax
    d\'ecimo%
    \or und\'ecimo%
    \or duod\'ecimo%
    \or decimotercero%
    \or decimocuarto%
    \or decimoquinto%
    \or decimosexto%
    \or decimos\'eptimo%
    \or decimoctavo%
    \or decimonoveno%
  \fi
}%
```

Hundreds:

```
\gdef\@@hundredthstringspanish#1{%
  \ifcase#1\relax
    \or cent\'esimo%
    \or ducent\'esimo%
    \or tricent\'esimo%
    \or cuadringent\'esimo%
    \or quingent\'esimo%
    \or sexcent\'esimo%
    \or septing\'esimo%
    \or octingent\'esimo%
    \or noningent\'esimo%
  \fi
}%
```

Units (feminine):

```
\gdef\@@unitthstringFspanish#1{%
  \ifcase#1\relax
    cera%
    \or primera%
    \or segunda%
    \or tercera%
    \or cuarta%
    \or quinta%
    \or sexta%
    \or s\'eptima%
    \or octava%
    \or novena%
  \fi
}%
```

Tens (feminine):

```
\gdef\@@tenthstringFspanish#1{%
  \ifcase#1\relax
    \or d\'ecima%
    \or vig\'esima%
    \or trig\'esima%
    \or cuadrag\'esima%
    \or quincuag\'esima%
    \or sexag\'esima%
    \or septuag\'esima%
    \or octog\'esima%
    \or nonag\'esima%
  \fi
}%
```

Teens (feminine)

```
\gdef\@@teenthstringFspanish#1{%
  \ifcase#1\relax
    d\'ecima%
    \or und\'ecima%
    \or duod\'ecima%
    \or decimotercera%
    \or decimocuarta%
    \or decimoquinta%
    \or decimosexta%
    \or decimos\'eptima%
    \or decimoctava%
    \or decimonovena%
  \fi
}%
```

Hundreds (feminine)

```
\gdef\@@hundredthstringFspanish#1{%
  \ifcase#1\relax
    \or cent\'esima%
    \or ducent\'esima%
    \or tricent\'esima%
    \or cuadringent\'esima%
    \or quingent\'esima%
    \or sexcent\'esima%
    \or septing\'esima%
    \or octingent\'esima%
    \or noningent\'esima%
  \fi
}%
```

As above, but with initial letters in upper case

```
\gdef\@@Unitthstringspanish#1{%
  \ifcase#1\relax
    Cero%
    \or Primero%
    \or Segundo%
    \or Tercero%
    \or Cuarto%
    \or Quinto%
    \or Sexto%
    \or S\'eptimo%
    \or Octavo%
    \or Noveno%
  \fi
}%
```

Tens:

```
\gdef\@@Tenthstringspanish#1{%
  \ifcase#1\relax
    \or D\'ecimo%
    \or Vig\'esimo%
    \or Trig\'esimo%
    \or Cuadrag\'esimo%
    \or Quincuag\'esimo%
    \or Sexag\'esimo%
    \or Septuag\'esimo%
    \or Octog\'esimo%
    \or Nonag\'esimo%
  \fi
}%
```

Teens:

```
\gdef\@@Teenthstringspanish#1{%
  \ifcase#1\relax
    D\'ecimo%
    \or Und\'ecimo%
    \or Duod\'ecimo%
    \or Decimotercero%
    \or Decimocuarto%
    \or Decimoquinto%
    \or Decimosexto%
    \or Decimos\'eptimo%
    \or Decimoctavo%
    \or Decimonoveno%
  \fi
}%
```

Hundreds

```
\gdef\@@Hundredthstringspanish#1{%
  \ifcase#1\relax
    \or Cent\'esimo%
    \or Ducent\'esimo%
    \or Tricent\'esimo%
    \or Cuadringent\'esimo%
    \or Quingent\'esimo%
    \or Sexcent\'esimo%
    \or Septing\'esimo%
    \or Octingent\'esimo%
    \or Noningent\'esimo%
  \fi
}%
```

As above, but feminine.

```
\gdef\@@UnitthstringFspanish#1{%
  \ifcase#1\relax
    Cera%
    \or Primera%
    \or Segunda%
    \or Tercera%
    \or Cuarta%
    \or Quinta%
    \or Sexta%
    \or S\'eptima%
    \or Octava%
    \or Novena%
  \fi
}%
```

Tens (feminine)

```
\gdef\@@TenthstringFspanish#1{%
  \ifcase#1\relax
    \or D\'ecima%
    \or Vig\'esima%
    \or Trig\'esima%
    \or Cuadrag\'esima%
    \or Quincuag\'esima%
    \or Sexag\'esima%
    \or Septuag\'esima%
    \or Octog\'esima%
    \or Nonag\'esima%
  \fi
}%
```

Teens (feminine):

```
\gdef\@@TeenthstringFspanish#1{%
  \ifcase#1\relax
    D\'ecima%
    \or Und\'ecima%
    \or Duod\'ecima%
    \or Decimotercera%
    \or Decimocuarta%
    \or Decimoquinta%
    \or Decimosexta%
    \or Decimos\'eptima%
    \or Decimoctava%
    \or Decimonovena%
  \fi
}%
```

Hundreds (feminine):

```
\gdef\@@HundredthstringFspanish#1{%
  \ifcase#1\relax
    \or Cent\'esima%
    \or Ducent\'esima%
    \or Tricent\'esima%
    \or Cuadringent\'esima%
    \or Quingent\'esima%
    \or Sexcent\'esima%
    \or Septing\'esima%
    \or Octingent\'esima%
    \or Noningent\'esima%
  \fi
}%
```

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

```
\gdef\@@numberstringspanish#1#2{%
\ifnum#1>99999
\PackageError{fmtcount}{Out of range}%
{This macro only works for values less than 100000}%
\else
\ifnum#1<0
\PackageError{fmtcount}{Negative numbers not permitted}%
{This macro does not work for negative numbers, however
you can try typing "minus" first, and then pass the modulus of
this number}%
\fi
\fi
\def#2{}%
\@strctr=#1\relax \divide\@strctr by 1000\relax
\ifnum\@strctr>9
```

#1 is greater or equal to 10000

```
  \divide\@strctr by 10
  \ifnum\@strctr>1
    \let\@@fc@numstr#2\relax
    \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
    \@strctr=#1 \divide\@strctr by 1000\relax
    \@modulo{\@strctr}{10}%
    \ifnum\@strctr>0\relax
       \let\@@fc@numstr#2\relax
       \edef#2{\@@fc@numstr\ \@andname\ \@unitstring{\@strctr}}%
    \fi
  \else
```

```
    \@strctr=#1\relax
    \divide\@strctr by 1000\relax
    \@modulo{\@strctr}{10}%
    \let\@@fc@numstr#2\relax
    \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
  \fi
  \let\@@fc@numstr#2\relax
  \edef#2{\@@fc@numstr\ \@thousand}%
\else
  \ifnum\@strctr>0\relax
    \ifnum\@strctr>1\relax
       \let\@@fc@numstr#2\relax
       \edef#2{\@@fc@numstr\@unitstring{\@strctr}\ }%
    \fi
    \let\@@fc@numstr#2\relax
    \edef#2{\@@fc@numstr\@thousand}%
  \fi
\fi
\@strctr=#1\relax \@modulo{\@strctr}{1000}%
\divide\@strctr by 100\relax
\ifnum\@strctr>0\relax
  \ifnum#1>1000\relax
    \let\@@fc@numstr#2\relax
    \edef#2{\@@fc@numstr\ }%
  \fi
  \@tmpstrctr=#1\relax
  \@modulo{\@tmpstrctr}{1000}%
  \ifnum\@tmpstrctr=100\relax
    \let\@@fc@numstr#2\relax
    \edef#2{\@@fc@numstr\@tenstring{10}}%
  \else
    \let\@@fc@numstr#2\relax
    \edef#2{\@@fc@numstr\@hundredstring{\@strctr}}%
  \fi
\fi
\@strctr=#1\relax \@modulo{\@strctr}{100}%
\ifnum#1>100\relax
  \ifnum\@strctr>0\relax
    \let\@@fc@numstr#2\relax
    \edef#2{\@@fc@numstr\ }%
  \fi
\fi
\ifnum\@strctr>29\relax
  \divide\@strctr by 10\relax
  \let\@@fc@numstr#2\relax
  \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
  \@strctr=#1\relax \@modulo{\@strctr}{10}%
  \ifnum\@strctr>0\relax
    \let\@@fc@numstr#2\relax
    \edef#2{\@@fc@numstr\ \@andname\ \@unitstring{\@strctr}}%
  \fi
\else
  \ifnum\@strctr<10\relax
    \ifnum\@strctr=0\relax
      \ifnum#1<100\relax
        \let\@@fc@numstr#2\relax
        \edef#2{\@@fc@numstr\@unitstring{\@strctr}}%
      \fi
    \else
      \let\@@fc@numstr#2\relax
      \edef#2{\@@fc@numstr\@unitstring{\@strctr}}%
    \fi
  \else
    \ifnum\@strctr>19\relax
      \@modulo{\@strctr}{10}%
      \let\@@fc@numstr#2\relax
      \edef#2{\@@fc@numstr\@twentystring{\@strctr}}%
    \else
      \@modulo{\@strctr}{10}%
      \let\@@fc@numstr#2\relax
      \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
    \fi
  \fi
\fi
}%
```

As above, but for ordinals

```
\gdef\@@ordinalstringspanish#1#2{%
\@strctr=#1\relax
\ifnum#1>99999
\PackageError{fmtcount}{Out of range}%
{This macro only works for values less than 100000}%
\else
\ifnum#1<0
\PackageError{fmtcount}{Negative numbers not permitted}%
{This macro does not work for negative numbers, however
you can try typing "minus" first, and then pass the modulus of
this number}%
\else
\def#2{}%
\ifnum\@strctr>999\relax
  \divide\@strctr by 1000\relax
  \ifnum\@strctr>1\relax
    \ifnum\@strctr>9\relax
      \@tmpstrctr=\@strctr
      \ifnum\@strctr<20
        \@modulo{\@tmpstrctr}{10}%
        \let\@@fc@ordstr#2\relax
        \edef#2{\@@fc@ordstr\@teenthstring{\@tmpstrctr}}%
      \else
        \divide\@tmpstrctr by 10\relax
        \let\@@fc@ordstr#2\relax
        \edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
        \@tmpstrctr=\@strctr
        \@modulo{\@tmpstrctr}{10}%
        \ifnum\@tmpstrctr>0\relax
          \let\@@fc@ordstr#2\relax
          \edef#2{\@@fc@ordstr\@unitthstring{\@tmpstrctr}}%
        \fi
      \fi
    \else
       \let\@@fc@ordstr#2\relax
       \edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
    \fi
  \fi
  \let\@@fc@ordstr#2\relax
  \edef#2{\@@fc@ordstr\@thousandth}%
\fi
\@strctr=#1\relax
\@modulo{\@strctr}{1000}%
\ifnum\@strctr>99\relax
  \@tmpstrctr=\@strctr
  \divide\@tmpstrctr by 100\relax
  \ifnum#1>1000\relax
    \let\@@fc@ordstr#2\relax
    \edef#2{\@@fc@ordstr\ }%
  \fi
  \let\@@fc@ordstr#2\relax
  \edef#2{\@@fc@ordstr\@hundredthstring{\@tmpstrctr}}%
\fi
\@modulo{\@strctr}{100}%
\ifnum#1>99\relax
  \ifnum\@strctr>0\relax
    \let\@@fc@ordstr#2\relax
    \edef#2{\@@fc@ordstr\ }%
  \fi
\fi
\ifnum\@strctr>19\relax
  \@tmpstrctr=\@strctr
  \divide\@tmpstrctr by 10\relax
  \let\@@fc@ordstr#2\relax
  \edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
  \@tmpstrctr=\@strctr
  \@modulo{\@tmpstrctr}{10}%
  \ifnum\@tmpstrctr>0\relax
    \let\@@fc@ordstr#2\relax
    \edef#2{\@@fc@ordstr\ \@unitthstring{\@tmpstrctr}}%
  \fi
```

```
\else
  \ifnum\@strctr>9\relax
    \@modulo{\@strctr}{10}%
    \let\@@fc@ordstr#2\relax
    \edef#2{\@@fc@ordstr\@teenthstring{\@strctr}}%
  \else
    \ifnum\@strctr=0\relax
      \ifnum#1=0\relax
        \let\@@fc@ordstr#2\relax
        \edef#2{\@@fc@ordstr\@unitstring{0}}%
      \fi
    \else
      \let\@@fc@ordstr#2\relax
      \edef#2{\@@fc@ordstr\@unitthstring{\@strctr}}%
    \fi
  \fi
\fi
\fi
\fi
}%
```

9.4.14 fc-UKenglish.def

English definitions

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

Loaded fc-english.def if not already loaded

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

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

```
\global\let\@ordinalMUKenglish\@ordinalMenglish
\global\let\@ordinalFUKenglish\@ordinalMenglish
\global\let\@ordinalNUKenglish\@ordinalMenglish
\global\let\@numberstringMUKenglish\@numberstringMenglish
\global\let\@numberstringFUKenglish\@numberstringMenglish
\global\let\@numberstringNUKenglish\@numberstringMenglish
\global\let\@NumberstringMUKenglish\@NumberstringMenglish
\global\let\@NumberstringFUKenglish\@NumberstringMenglish
\global\let\@NumberstringNUKenglish\@NumberstringMenglish
\global\let\@ordinalstringMUKenglish\@ordinalstringMenglish
\global\let\@ordinalstringFUKenglish\@ordinalstringMenglish
\global\let\@ordinalstringNUKenglish\@ordinalstringMenglish
\global\let\@OrdinalstringMUKenglish\@OrdinalstringMenglish
\global\let\@OrdinalstringFUKenglish\@OrdinalstringMenglish
\global\let\@OrdinalstringNUKenglish\@OrdinalstringMenglish
```

9.4.15 fc-USenglish.def

US English definitions

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

Loaded fc-english.def if not already loaded

```
\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.)

```
\global\let\@ordinalMUSenglish\@ordinalMenglish
\global\let\@ordinalFUSenglish\@ordinalMenglish
\global\let\@ordinalNUSenglish\@ordinalMenglish
\global\let\@numberstringMUSenglish\@numberstringMenglish
\global\let\@numberstringFUSenglish\@numberstringMenglish
\global\let\@numberstringNUSenglish\@numberstringMenglish
\global\let\@NumberstringMUSenglish\@NumberstringMenglish
\global\let\@NumberstringFUSenglish\@NumberstringMenglish
\global\let\@NumberstringNUSenglish\@NumberstringMenglish
\global\let\@ordinalstringMUSenglish\@ordinalstringMenglish
\global\let\@ordinalstringFUSenglish\@ordinalstringMenglish
\global\let\@ordinalstringNUSenglish\@ordinalstringMenglish
\global\let\@OrdinalstringMUSenglish\@OrdinalstringMenglish
\global\let\@OrdinalstringFUSenglish\@OrdinalstringMenglish
\global\let\@OrdinalstringNUSenglish\@OrdinalstringMenglish
```