

fmtcount.sty: Displaying the Values of \LaTeX Counters

Nicola L.C. Talbot

Vincent Belaïche

www.dickimaw-books.com

2012-10-24 (version 2.02)

Contents

1	Introduction	2
2	Available Commands	2
3	Package Options	8
4	Multilingual Support	8
4.1	Options for French	9
4.2	Prefixes	14
5	Configuration File <code>fmtcount.cfg</code>	14
6	LaTeX2HTML style	14
7	Acknowledgements	15
8	Troubleshooting	15
9	The Code	15
9.1	<code>fcnumparser.sty</code>	15
9.2	<code>fcprefix.sty</code>	25
9.3	<code>fmtcount.sty</code>	36
9.4	Multilanguage Definitions	59
9.4.1	<code>fc-american.def</code>	62
9.4.2	<code>fc-british.def</code>	63
9.4.3	<code>fc-english.def</code>	63
9.4.4	<code>fc-francais.def</code>	73
9.4.5	<code>fc-french.def</code>	74
9.4.6	<code>fc-frenchb.def</code>	104
9.4.7	<code>fc-german.def</code>	104

9.4.8	<code>fc-germanb.def</code>	113
9.4.9	<code>fc-italian</code>	114
9.4.10	<code>fc-ngerman.def</code>	115
9.4.11	<code>fc-ngermanb.def</code>	115
9.4.12	<code>fc-portuges.def</code>	115
9.4.13	<code>fc-spanish.def</code>	129
9.4.14	<code>fc-UKenglish.def</code>	145
9.4.15	<code>fc-USenglish.def</code>	146

1 Introduction

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

2 Available Commands

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

`\ordinal`

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

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

`\fmtord`

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

is used to format the st, nd, rd, th bit. By default the ordinal is formatted as a superscript, if the package option `level` is used, it is level with the text. For example, if the current section is 3, then `\ordinal{section}` will produce the output: 3rd. Note that the optional argument `<gender>` occurs *at the end*. This argument may only take one of the following values: `m` (masculine), `f` (feminine) or `n` (neuter.) If `<gender>` is omitted, or if the given gender has no meaning in the current language, `m` is assumed.

Notes:

1. the memoir class also defines a command called `\ordinal` which takes a number as an argument instead of a counter. In order to overcome this incompatibility, if you want to use the `fmtcount` package with the memoir class you should use

`\FCordinal`

```
\FCordinal
```

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

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

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

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

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

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

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

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

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

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

This does the same as `\numberstring`, but converts the string to upper case. Note that `\MakeUppercase{\NUMBERstring{<counter>}}` doesn't work, due to the way that `\MakeUppercase` expands its argument¹.

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

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

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

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.

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

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

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

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

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

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

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

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

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

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

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

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

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

`\FMCuse` `\FMCuse{<label>}`

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

\storeordinal	\storeordinal{<label>}{<counter>}[<gender>]
\storeordinalstring	\storeordinalstring{<label>}{<counter>}[<gender>]
\storeOrdinalstring	\storeOrdinalstring{<label>}{<counter>}[<gender>]
\storeORDINALstring	\storeORDINALstring{<label>}{<counter>}[<gender>]
\storenumberstring	\storenumberstring{<label>}{<counter>}[<gender>]
\storeNumberstring	\storeNumberstring{<label>}{<counter>}[<gender>]
\storeNUMBERstring	\storeNUMBERstring{<label>}{<counter>}[<gender>]
\storeordinalnum	\storeordinalnum{<label>}{<number>}[<gender>]
\storeordinalstringnum	\storeordinalstring{<label>}{<number>}[<gender>]
\storeOrdinalstringnum	\storeOrdinalstringnum{<label>}{<number>}[<gender>]
\storeORDINALstringnum	\storeORDINALstringnum{<label>}{<number>}[<gender>]
\storenumberstringnum	\storenumberstring{<label>}{<number>}[<gender>]
\storeNumberstringnum	\storeNumberstring{<label>}{<number>}[<gender>]
\storeNUMBERstringnum	\storeNUMBERstring{<label>}{<number>}[<gender>]

`\binary` `\binary{<counter>}`

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

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

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

`\binarynum` `\binary{<n>}`

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

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

`\octal` `\octal{<counter>}`

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

`\octalnum` `\octalnum{<n>}`

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

`\hexadecimal` `\hexadecimal{<counter>}`

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

`\Hexadecimal` `\Hexadecimal{<counter>}`

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

`\hexadecimalnum` `\hexadecimalnum{<n>}`

`\Hexadecimalnum` `\Hexadecimalnum{<n>}`

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

`\decimal`

`\decimal{<counter>}`

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

`\decimalnum`

`\decimalnum{<n>}`

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

`\aaalph`

`\aaalph{<counter>}`

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

`\AAAlph`

`\AAAlph{<counter>}`

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

`\aaalphnum`

`\aaalphnum{<n>}`

`\AAAlphnum`

`\AAAlphnum{<n>}`

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

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

`\abalph`

`\abalph{<counter>}`

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

`\ABAlph`

`\ABAlph{<counter>}`

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

will produce: DU.

`\abalphnum`

`\abalphnum{<n>}`

`\ABAlphnum`

`\ABAlphnum{<n>}`

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

3 Package Options

The following options can be passed to this package:

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

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

These can also be set using the command:

`\fmtcountsetoptions`

`\fmtcountsetoptions{fmtord=<type>}`

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

4 Multilingual Support

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

The package checks to see if the command `\l@<language>` is defined⁴, and will load the code for those languages. The commands `\ordinal`, `\ordinalstring` and `\numberstring` (and their variants) will then be formatted in the currently selected language.

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

The male gender for all languages is used by default, however the feminine or neuter forms can be obtained by passing `f` or `n` as an optional argument to `\ordinal`, `\ordinalnum` etc. For example: `\numberstring{section}[f]`. Note that the optional argument comes *after* the compulsory argument. If a

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

³Thanks to Edoardo Pasca for supplying the information.

⁴this will be true if you have loaded `babel`

gender is not defined in a given language, the masculine version will be used instead.

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

4.1 Options for French

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

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

`\fmtcountsetoptions`

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

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

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

`dialect`

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

`french`

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

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

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

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

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

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

Il est à noter que la variante `belgian` est parfaitement correcte pour les francophones français⁶, et qu'elle est également utilisée en Suisse Romande hormis dans les cantons de Vaud, du Valais et de Fribourg. En ce qui concerne le mot “octante”, il n'est actuellement pas pris en charge et n'est guère plus utilisé, ce qui est sans doute dommage car il est sans doute plus acceptable que le “huitante” de certains de nos amis suisses.

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

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

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

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

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

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

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

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

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

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

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

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

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

- la règle de l'accord en nombre des noms de nombre dans un numéral cardinal dépend de savoir s'il a vraiment une valeur cardinale ou bien une valeur ordinale, ainsi on écrit « aller à la page deux-cent (sans s) d'un livre de deux-cents (avec s) pages », il faut donc pouvoir changer la configuration pour sélectionner le cas considéré,
- un autre cas demandant quelque configurabilité est celui de « mil » et « mille ». Pour rappel « mille » est le pluriel irrégulier de « mil », mais l'alter-

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

- finalement les règles du français quoique bien définies ne sont pas très cohérentes et il est donc inévitable qu’un jour ou l’autre on les simplifie. Le paquetage `fmtcount` est déjà prêt à cette éventualité.

Le paramètre *⟨french plural control⟩* peut prendre les valeurs suivantes :

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

multiple l-last	pour marquer le pluriel lorsque le nombre considéré est multiplié par au moins 2 et est <i>localement</i> en dernière position, où “localement” signifie qu’on considère seulement la portion du nombre qui multiplie soit l’unité, soit un $\langle n \rangle$ illion ou un $\langle n \rangle$ illiard ; ceci est la convention en vigueur pour le pluriel de “vingt” et de “cent” lorsque le nombre formaté a une valeur cardinale,
multiple lng-last	pour marquer le pluriel lorsque le nombre considéré est multiplié par au moins 2 et est <i>localement</i> mais <i>non globalement</i> en dernière position, où “localement” et <i>globalement</i> ont la même signification que pour les options multiple g-last et multiple l-last ; ceci est la convention en vigueur pour le pluriel de “vingt” et de “cent” lorsque le nombre formaté a une valeur ordinale,
multiple ng-last	pour marquer le pluriel lorsque le nombre considéré est multiplié par au moins 2, et <i>n</i> ’est pas <i>globalement</i> en dernière position, où “globalement” a la même signification que pour l’option multiple g-last ; ceci est la règle que j’infère être en vigueur pour les nombres de la forme $\langle n \rangle$ illion et $\langle n \rangle$ illiard lorsque le nombre a une valeur ordinale, mais à dire vrai pour des nombres aussi grands, par exemple « deux millions », je pense qu’il n’est tout simplement pas d’usage de dire « l’exemplaire deux million(s ?) » pour « le deux millionième exemplaire ».

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

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

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

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

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

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

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

Par défaut, l’option vaut `reformed`.

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

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

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

Par défaut, l’option vaut `recursive`.

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

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

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

mil plural mark `\fmtcountsetoptions{french={mil plural mark=<any text>}}`

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

4.2 Prefixes

`\latinnumeralstring`

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

`\latinnumeralstringnum`

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

5 Configuration File `fmtcount.cfg`

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

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

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

and if `fmtcount.cfg` has the line:

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

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

6 LaTeX2HTML style

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

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

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

This, I agree, is an unpleasant cludge.

7 Acknowledgements

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

8 Troubleshooting

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

9 The Code

9.1 fcnumparser.sty

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

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

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

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

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

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

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

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

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

No comma either, so fractional part is set empty.

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

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

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

```

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

\fc @number@parser Macro \fc@number@parser is the main engine to parse a
number. Argument '#1' is input and contains the number to be parsed. At end
of this macro, each digit is stored separately in a \fc@digit@<n>, and macros
\fc@min@weight and \fc@max@weight are set to the bounds for <n>.
26 \def\fc@number@parser#1{%
First remove all the spaces in #1, and place the result into \@tempa.
27 \let\@tempa\@empty
28 \def\@tempb##1##2\fc@nil{%
29 \def\@tempc{##1}%
30 \ifx\@tempc\space
31 \else
32 \expandafter\def\expandafter\@tempa\expandafter{\@tempa ##1}%
33 \fi
34 \def\@tempc{##2}%
35 \ifx\@tempc\@empty
36 \expandafter\@gobble
37 \else
38 \expandafter\@tempb
39 \fi
40 ##2\fc@nil
41 }%
42 \@tempb#1\fc@nil

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

```


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

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

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

```
61 \expandafter\fc@digit@counter\fc@unit@weight
```

```
62 \expandafter\fc@parse@integer@digits\fc@integer@part\fc@end\fc@nil
```

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

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

```
63 \fc@digit@counter -\fc@digit@counter
```

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

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

```
66 \advance\fc@digit@counter by -1 %
```

```
67 \edef\fc@max@weight{\the\fc@digit@counter}%
```

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

```
68 {%
```

```
69 \count0 \fc@unit@weight\relax
```

```
70 \count1 \fc@max@weight\relax
```

```
71 \advance\count0 by -\count1 %
```

```
72 \advance\count0 by -1 %
```

```
73 \def\@tempa##1{\def\@tempb{\def\@tempi{##1}}}%
```

```
74 \expandafter\@tempa\expandafter{\the\count0}%
```

```
75 \expandafter
```

```
76 }\@tempb
```

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

```
77 \expandafter\fc@digit@counter\fc@unit@weight
```

```
78 \def\@templ{%
```

```
79 \ifnum\fc@digit@counter>\fc@max@weight
```

```
80 \let\next\relax
```

```
81 \else
```

Here is the loop body:

```
82 {%
```

```
83 \count0 \@tempi
```

```
84 \advance\count0 by \fc@digit@counter
```

```
85 \expandafter\def\expandafter\@tempd\expandafter{\csname fc@digit@\the\count0\endcsname}
```

```
86 \expandafter\def\expandafter\@tempe\expandafter{\csname fc@digit@\the\fc@digit@counter\endcsname}
```

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

```
88 \expandafter\expandafter\expandafter\@tempa\expandafter\@tempe\@tempd
```

```

89         \expandafter
90     } \@tempb
91     \advance\fc@digit@counter by 1 %
92     \fi
93     \next
94 }%
95 \let\next\@templ
96 \@templ

```

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

```

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

```

```

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

```

Now we have macros to read a few digits from the $\text{\fc@digit@}\langle n \rangle$ array and form a corresponding number.

```

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

```

Arguments as follows:

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

```

121 \def\fc@read@unit#1#2{%

```

```

122 \ifnum#2>\fc@max@weight
123     #1=0\relax
124 \else
125     \ifnum#2<\fc@min@weight
126         #1=0\relax
127     \else
128         {%
129             \edef\@tempa{\number#2}%
130             \count0=\@tempa
131             \edef\@tempa{\csname fc@digit@the\count0\endcsname}%
132             \def\@tempb##1{\def\@tempa{#1=##1\relax}}%
133             \expandafter\@tempb\expandafter{\@tempa}%
134             \expandafter
135         }\@tempa
136     \fi
137 \fi
138 }

```

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

```

139 \ifundefined{fc@read@hundred}{%}%
140 \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro 'fc@read@hundred'}%

```

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

#1 output counter: into which the read value is placed

#2 input number: unit weight at which reach the value is to be read

```

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

```

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

```

155 \ifundefined{fc@read@thousand}{%}%
156 \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro
157     'fc@read@thousand'}}%

```

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

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

```

158 \def\fc@read@thousand#1#2{%
159   {%
160     \fc@read@unit{\count0}{#2}%
161     \def\@tempa##1{\fc@read@hundred{\count1}{##1}}%
162     \count2=#2%
163     \advance\count2 by 1 %
164     \expandafter\@tempa{\the\count2}%
165     \multiply\count1 by 10 %
166     \advance\count1 by \count0 %
167     \def\@tempa##1{\def\@tempb{#1=##1\relax}}
168     \expandafter\@tempa\expandafter{\the\count1}%
169     \expandafter
170   }\@tempb
171 }

```

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

First we check that the macro is not yet defined.

```

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

```

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

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

```

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

```

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

```

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

```

Arguments as follows:

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

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

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

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

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

Then we do the real job

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

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

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

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

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

```
204 \def\fc@@check@nonzeros@inner{%
205   \ifnum\count0<\fc@min@weight
206     \count0=\fc@min@weight\relax
207   \fi
208   \ifnum\count1>\fc@max@weight\relax
209     \count1=\fc@max@weight
```

```

210 \fi
211 \count2\count0 %
212 \advance\count2 by 1 %
213 \ifnum\count0>\count1 %
214   \PackageError{fcnumparser}{Unexpected arguments}{Number in argument 2 of macro
215     'fc@check@nonzeros' must be at least equal to number in argument 1}%
216 \else
217   \fc@@check@nonzeros@inner@loopbody
218   \ifnum\@tempc>0 %
219     \ifnum\@tempc<9 %
220       \ifnum\count0>\count1 %
221       \else
222         \let\@tempd\@tempc
223         \fc@@check@nonzeros@inner@loopbody
224         \ifnum\@tempc=0 %
225           \let\@tempc\@tempd
226         \else
227           \def\@tempc{9}%
228         \fi
229       \fi
230     \fi
231   \fi
232 \fi
233 }
234 \def\fc@@check@nonzeros@inner@loopbody{%
235   % \@tempc <- digit of weight \count0
236   \expandafter\let\expandafter\@tempc\csname fc@digit@\the\count0\endcsname
237   \advance\count0 by 1 %
238   \ifnum\@tempc=0 %
239     \ifnum\count0>\count1 %
240       \let\next\relax
241     \else
242       \let\next\fc@@check@nonzeros@inner@loopbody
243     \fi
244   \else
245     \ifnum\count0>\count2 %
246       \def\@tempc{9}%
247     \fi
248     \let\next\relax
249   \fi
250   \next
251 }
\fc @intpart@find@last Macro \fc@intpart@find@last find the rightmost non
zero digit in the integer part. First check that the macro is not yet defined.
252 \@ifundefined{fc@intpart@find@last}{\{%
253   \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro
254     'fc@intpart@find@last'}}

```

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

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

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

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

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

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

Now propagate result after closing bracket into counter #1.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

\@tempb is not empty, first we use

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

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

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

```
313 \@ifundefined{fc@get@last@letter}{-}{\PackageError{fcnumparser}{Duplicate definition}{Redefi
```



```

314   of macro 'fc@get@last@letter'}}}%
315 \def\fc@get@last@letter#1#2#3{%
316   {%

```

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

```

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

```

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

```

320   \def\@tempa##1##2\fc@nil{%
321     \toks2{##1}%
322     \toks3{##2}%
323     \def\@tempb{##2}%
324   }%

```

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

```

325   \def\@templ{%
326     \expandafter\@tempa\the\toks1 \fc@nil
327     \ifx\@tempb\@empty

```

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

```

328       \let\next\relax
329     \else

```

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

```

330       \expandafter\expandafter\expandafter\@tempt
331       \expandafter\expandafter\expandafter{%
332         \expandafter\the\expandafter\@tempt
333         \the\toks2}%

```

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

```

334       \toks1\toks3 %
335       \fi
336       \next
337     }%

```

Here run the loop.

```

338     \let\next\@templ
339   \next

```

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

```

340   \edef\@tempa{\def\noexpand#2{\the\toks0}\def\noexpand#3{\the\toks1}}}%
341   \expandafter
342   }\@tempa
343 }%

```

9.2 fcprefix.sty

Pseudo-latin prefixes.

```

344 \NeedsTeXFormat{LaTeX2e}
345 \ProvidesPackage{fcprefix}[2012/09/28]

```

```

346 \RequirePackage{ifthen}
347 \RequirePackage{keyval}
348 \RequirePackage{fcnumparser}

```

Option ‘use duode and unde’ is to select whether 18 and suchlikes ($\langle x \rangle 8$, $\langle x \rangle 9$) writes like duodevicies, or like octodecies. For French it should be ‘below 20’. Possible values are ‘below 20’ and ‘never’.

```

349 \define@key{fcprefix}{use duode and unde}[below20]{%
350   \ifthenelse{\equal{#1}{below20}}{%
351     \def\fc@duodeandunde{2}%
352   }{%
353     \ifthenelse{\equal{#1}{never}}{%
354       \def\fc@duodeandunde{0}%
355     }{%
356       \PackageError{fcprefix}{Unexpected option}{%
357         Option ‘use duode and unde’ expects ‘below 20’ or ‘never’ }%
358     }%
359   }%
360 }

```

Default is ‘below 20’ like in French.

```

361 \def\fc@duodeandunde{2}

```

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

```

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

```

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

```

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

```

```

384 }%
385 }

```

Default is to set accute accent like in French.

```

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

```

Option ‘power of millia’ tells how millia is raise to power n. It expects value:

recursive for which millia squared is noted as ‘milliamillia’

arabic for which millia squared is noted as ‘millia²’

prefix for which millia squared is noted as ‘bismillia’

```

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

```

Arguments as follows:

#1 output macro

#2 number with current weight w

```

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

```

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

#1 output macro

#2 number with current weight w

```

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

```

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

```

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

```

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

```

413   \count0=0 %
414   \let\@tempa\@empty
415   \loop

```

```

416      \ifnum\count0<\@tempb
417      \advance\count0 by 1 %
418      \expandafter\def
419      \expandafter\@tempa\expandafter{\@tempa millia}%
420  \repeat

```

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

```

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

```

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

#1 output macro
#2 number with current weight w

```

425 \def\fc@@arabic@millia#1#2{%
426   \ifnnum#2=0 %
427   \let#1\@empty
428   \else
429   \edef#1{millia\^{}\the#2}%
430   \fi
431 }

```

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

#1 output macro
#2 number with current weight w

```

432 \def\fc@@prefix@millia#1#2{%
433   \fc@@latin@numeral@prefix{#2}{#1}%
434 }

```

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

```

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

```

\fc @@latin@cardinal@prefix Compute a cardinal prefix for n-illion, like $1 \Rightarrow$ ‘m’, $2 \Rightarrow$ ‘bi’, $3 \Rightarrow$ ‘tri’. The algorithm to derive this prefix is that of Russ Rowlett I founds its documentation on Alain Lassine’s site: http://www.alain.be/Boece/grands_nombres.html. First check that macro is not yet defined.

```

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

```

Arguments as follows:

#1 input number to be formatted
#2 output macro name into which to place the formatted result

```

439 \def\fc@@latin@cardinal@prefix#1#2{%
440   {%

```

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

```

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

```

Now parse number from expanded input.

```

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

```

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

```

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

```

\@tempm will hold the millia^{n÷3}

```

446 \let\@tempm\@empty

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

468       \ifcase\count5 %
469       \or cen%
470       \or ducen%
471       \or trecen%
472       \or quadringen%
473       \or quingen%
474       \or sescen%
475       \or septigen%

```

```

476         \or octingen%
477         \or nongen%
478     \fi

```

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

```

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

```

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

```

495     \ifcase\count3 %
496     \or \ifnum\count2>\fc@max@weight\else un\fi
497     \or d\fc@u@in@duo o%
498     \or tre%
499     \or quattuor%
500     \or quin%
501     \or sex%
502     \or septen%
503     \or octo%
504     \or novem%
505     \fi
506 \fi
507 \else
508     % x(10..99)
509     \ifcase\count3 %
510     \or un%
511     \or d\fc@u@in@duo o%
512     \or tre%
513     \or quattuor%
514     \or quin%
515     \or sex%
516     \or septen%
517     \or octo%
518     \or novem%
519     \fi
520 \ifcase\count4 %

```

```

521         \or dec%
522         \or vigin\@tempt
523         \or trigin\@tempt
524         \or quadragin\@tempt
525         \or quinquagin\@tempt
526         \or sexagin\@tempt
527         \or septuagin\@tempt
528         \or octogin\@tempt
529         \or nonagin\@tempt
530     \fi
531 \fi

```

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

```

532     \@tempm

```

And append previous version of \@tempa.

```

533     \@temp
534 }%

```

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

```

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

```

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

```

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

```

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

```

547 \ifundefined{fc@@latin@numeral@pexif}{%
548   \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
549     ‘fc@@latin@numeral@pexif’}%

```

Arguments as follows:

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

```

550 \def\fc@@latin@numeral@pexif#1#2{%
551   {%

```

```

552 \edef\@tempa{\number#1}%
553 \def\fc@unit@weight{0}%
554 \expandafter\fc@number@parser\expandafter{\@tempa}%
555 \count2=0 %

```

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

```

556 \let\@tempm\@empty

```

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

```

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

```

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

```

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

```

Check the use of duodevicies instead of octodecies.

```

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

```



```

593         \or septing%
594         \or octing%
595         \or nong%
596         \fi
597         {enties}%
598         \ifnum\count4=0 %

```

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

```

599         \ifcase\count3 %
600         \or
601         \ifnum\count2=3 %
602         s\fc@prefix@eacute emel%
603         \else
604         \ifnum\count2>\fc@max@weight\else un\fi
605         \fi
606         \or bis%
607         \or ter%
608         \or quater%
609         \or quinquies%
610         \or sexies%
611         \or septies%
612         \or octies%
613         \or novies%
614         \fi
615         \else

```

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

```

616         \ifcase\count3 %
617         \or un%
618         \or d\fc@u@in@duo o%
619         \or ter%
620         \or quater%
621         \or quin%
622         \or sex%
623         \or septen%
624         \or \@temps{octo}{duod\fc@prefix@eacute e}% x8 = two before next (x+1)0
625         \or \@temps{novem}{und\fc@prefix@eacute e}% x9 = one before next (x+1)0
626         \fi
627         \ifcase\count4 %
628         % can't get here
629         \or d\fc@prefix@eacute ec%
630         \or vic%
631         \or tric%
632         \or quadrag%
633         \or quinquag%
634         \or sexag%
635         \or septuag%
636         \or octog%
637         \or nonag%
638         \fi

```

```

639         ies%
640     \fi
641     % Insert the millies^(n/3) only if one of \count3 \count4 \count5 is non zero
642     \@tempm
643     % add up previous version of \@tempa
644     \@tempm
645 }%

```

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

```

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

```

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

```

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

```

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

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

```

659 \def\fc@call@opt@arg@second#1#2{%
660 \def\@tempb{%
661 \ifx[\@tempa
662 \def\@tempc[####1]{%
663     {#1{####1}{#2}}}%

```

```

664     }%
665     \else
666     \def\@tempc{\#1{\#2}}}%
667     \fi
668     \@tempc
669 }%
670 \futurelet\@tempa
671 \@tempb
672 }

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

```

User API.

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

- #1 local options
- #2 input number

```

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

```

Arguments as follows:

- #1 local options
- #2 input counter

```

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

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

```

```

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

```

9.3 fmtcount.sty

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

```

705 \NeedsTeXFormat{LaTeX2e}
706 \ProvidesPackage{fmtcount}[2012/10/24 v2.02]
707 \RequirePackage{ifthen}
708 \RequirePackage{keyval}
709 \RequirePackage{etoolbox}
710 \RequirePackage{fcprefix}

```

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

```

711 \RequirePackage{amsgen}

```

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

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

```
\fmtord
```

```

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

```

```
\padzeroes
```

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

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

```

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

```

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

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

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

```

716 \newcount\fc@tmpcatcode
717 \def\fc@languages{}
718 \def\fc@mainlang{}%
719 \newcommand*\FCloadlang}[1]{%
720   \@FC@iflangloaded{#1}{}%
721   {%
722     \fc@tmpcatcode=\catcode'\@ \relax
723     \catcode '\@ 11\relax
724     \input{fc-#1.def}%

```

```

725 \catcode '@ \fc@tmpcatcode\relax
726 \ifdefempty{\fc@languages}%
727 {%
728 \def\fc@languages{#1}%
729 }%
730 {%
731 \appto\fc@languages{,#1}%
732 }%
733 \def\fc@mainlang{#1}%
734 }%
735 }

```

\@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>`

```

736 \newcommand{\@FC@iflangloaded}[3]{%
737 \ifcsundef{ver@fc-#1.def}{#3}{#2}%
738 }

```

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

```

739 \newcommand*\ProvidesFCLanguage[1]{%
740 \ProvidesFile{fc-#1.def}%
741 }

```

\@fc@loadifbabelldf \@fc@loadigbabelldf{<language>}

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

```

742 \newcommand*\@fc@loadifbabelldf[1]{%
743 \ifcsundef{ver@#1.ldf}{\FCloadlang{#1}}%
744 }

```

Load appropriate language definition files:

```

745 \@fc@loadifbabelldf{english}
746 \@fc@loadifbabelldf{UKenglish}
747 \@fc@loadifbabelldf{british}
748 \@fc@loadifbabelldf{USenglish}
749 \@fc@loadifbabelldf{american}
750 \@fc@loadifbabelldf{spanish}
751 \@fc@loadifbabelldf{portuges}
752 \@fc@loadifbabelldf{french}
753 \@fc@loadifbabelldf{frenchb}

```

```

754 \@fc@loadifbabelldf{german}%
755 \@fc@loadifbabelldf{germanb}%
756 \@fc@loadifbabelldf{ngerman}%
757 \@fc@loadifbabelldf{ngermanb}%
758 \@fc@loadifbabelldf{italian}

```

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

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

`french`

```

760 \define@key{fmtcount}{french}[france]{%
761   \@ifundefined{datefrench}%
762   {%
763     \PackageError{fmtcount}%
764     {Language ‘french’ not defined}%
765     {You need to load babel before loading fmtcount}%
766   }%
767   {%
768     \setkeys{fcfrench}{#1}%
769   }%
770 }

```

`fmtord` Key to determine how to display the ordinal

```

771 \define@key{fmtcount}{fmtord}{%
772   \ifthenelse{\equal{#1}{level}
773               \or\equal{#1}{raise}
774               \or\equal{#1}{user}}%
775   {%
776     \def\fmtcount@fmtord{#1}%
777   }%
778   {%
779     \PackageError{fmtcount}%
780     {Invalid value ‘#1’ to fmtord key}%
781     {Option ‘fmtord’ can only take the values ‘level’, ‘raise’
782      or ‘user’}%
783   }%
784 }

```

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

```

785 \newif\iffmtord@abbrv
786 \fmtord@abbrvfalse
787 \define@key{fmtcount}{abbrv}[true]{%
788   \ifthenelse{\equal{#1}{true}\or\equal{#1}{false}}%
789   {%
790     \csname fmtord@abbrv#1\endcsname
791   }%

```

```

792  {%
793    \PackageError{fmtcount}%
794    {Invalid value ‘#1’ to fmtord key}%
795    {Option ‘fmtord’ can only take the values ‘true’ or
796     ‘false’}%
797  }%
798 }

```

prefix

```

799 \define@key{fmtcount}{prefix}[scale=long]{%
800   \RequirePackage{fmtprefix}%
801   \fmtprefixsetoption{#1}%
802 }

```

\fmtcountsetoptions Define command to set options.

```

803 \newcommand*{\fmtcountsetoptions}[1]{%
804   \def\fmtcount@fmtord{}%
805   \setkeys{fmtcount}{#1}%
806   \@ifundefined{datefrench}{}%
807   {%
808     \edef\@OrdinalstringMfrench{\noexpand
809       \csname @OrdinalstringMfrench\fmtcount@french\noexpand\endcsname}%
810     \edef\@OrdinalstringFfrench{\noexpand
811       \csname @OrdinalstringFfrench\fmtcount@french\noexpand\endcsname}%
812     \edef\@OrdinalstringMfrench{\noexpand
813       \csname @OrdinalstringMfrench\fmtcount@french\noexpand\endcsname}%
814     \edef\@OrdinalstringFfrench{\noexpand
815       \csname @OrdinalstringFfrench\fmtcount@french\noexpand\endcsname}%
816     \edef\@NumberstringMfrench{\noexpand
817       \csname @NumberstringMfrench\fmtcount@french\noexpand\endcsname}%
818     \edef\@NumberstringFfrench{\noexpand
819       \csname @NumberstringFfrench\fmtcount@french\noexpand\endcsname}%
820     \edef\@NumberstringMfrench{\noexpand
821       \csname @NumberstringMfrench\fmtcount@french\noexpand\endcsname}%
822     \edef\@NumberstringFfrench{\noexpand
823       \csname @NumberstringFfrench\fmtcount@french\noexpand\endcsname}%
824   }%
825   \ifthenelse{\equal{\fmtcount@fmtord}{level}}{%
826     {%
827       \renewcommand{\fmtord}[1]{##1}%
828     }%
829     {%
830       \ifthenelse{\equal{\fmtcount@fmtord}{raise}}{%
831         {%
832           \renewcommand{\fmtord}[1]{\textsuperscript{##1}}%
833         }%
834         {%
835           }%
836       }

```

```
837 }
```

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

```
838 \InputIfFileExists{fmtcount.cfg}%
839 {%
840   \PackageInfo{fmtcount}{Using configuration file fmtcount.cfg}%
841 }%
842 {%
843 }
```

level

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

raise

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

Process package options

```
848 \ProcessOptions
```

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

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

```
849 \newcount\@DT@modctr
850 \def\@modulo#1#2{%
851   \@DT@modctr=#1\relax
852   \divide \@DT@modctr by #2\relax
853   \multiply \@DT@modctr by #2\relax
854   \advance #1 by -\@DT@modctr
855 }
```

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

```
856 \newcount\@ordinalctr
857 \newcount\@orgargctr
858 \newcount\@strctr
859 \newcount\@tmpstrctr
```

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

```
860 \newif\if@DT@padzeroes
861 \newcount\@DT@loopN
862 \newcount\@DT@X
```


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

```

863 \newcommand*{\@binary}[1]{%
864   \@DT@padzeroestru
865   \@DT@loopN=17\relax
866   \@strctr=\@DT@loopN
867   \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by 1}%
868   \@strctr=65536\relax
869   \@DT@X=#1\relax
870   \loop
871     \@DT@modctr=\@DT@X
872     \divide\@DT@modctr by \@strctr
873     \ifthenelse{\boolean{\@DT@padzeroes}
874       \and \(\@DT@modctr=0\)}
875       \and \(\@DT@loopN>\c@padzeroesN\)}%
876     {}%
877     {\the\@DT@modctr}%
878     \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
879     \multiply\@DT@modctr by \@strctr
880     \advance\@DT@X by -\@DT@modctr
881     \divide\@strctr by 2\relax
882     \advance\@DT@loopN by -1\relax
883   \ifnum\@strctr>1
884     \repeat
885     \the\@DT@X
886 }
887
888 \let\binarynum=\@binary

```

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

```

889 \newcommand*{\@octal}[1]{%
890   \ifnum#1>32768
891     \PackageError{fmtcount}%
892     {Value of counter too large for \protect\@octal}
893     {Maximum value 32768}
894   \else
895     \@DT@padzeroestru
896     \@DT@loopN=6\relax
897     \@strctr=\@DT@loopN
898     \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by 1}%
899     \@strctr=32768\relax
900     \@DT@X=#1\relax
901     \loop
902       \@DT@modctr=\@DT@X
903       \divide\@DT@modctr by \@strctr
904       \ifthenelse{\boolean{\@DT@padzeroes}
905         \and \(\@DT@modctr=0\)}
906         \and \(\@DT@loopN>\c@padzeroesN\)}%
907       {}{\the\@DT@modctr}%
908     \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi

```

```

909    \multiply\@DT@modctr by \@strctr
910    \advance\@DT@X by -\@DT@modctr
911    \divide\@strctr by 8\relax
912    \advance\@DT@loopN by -1\relax
913    \ifnum\@strctr>1
914    \repeat
915    \the\@DT@X
916    \fi
917}
918\let\octalnum=\@octal

```

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

```

919\newcommand*\@@hexadecimal}[1]{%
920  \ifcase#1\or1\or2\or3\or4\or5\or
921  6\or7\or8\or9\or a\or b\or c\or d\or e\or f\fi
922}

```

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

```

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

```

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

```

947\newcommand*\@@Hexadecimal}[1]{%
948  \ifcase#1\or1\or2\or3\or4\or5\or6\or
949  7\or8\or9\or A\or B\or C\or D\or E\or F\fi
950}

```

\Hexadecimalnum Uppercase hexadecimal

```
951 \newcommand*{\@Hexadecimal}[1]{%
952   \@DT@padzeroes
953   \@DT@loopN=5\relax
954   \@strctr=\@DT@loopN
955   \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by 1}%
956   \@strctr=65536\relax
957   \@DT@X=#1\relax
958   \loop
959     \@DT@modctr=\@DT@X
960     \divide\@DT@modctr by \@strctr
961     \ifthenelse{\boolean{\@DT@padzeroes}
962       \and \(\@DT@modctr=0\)}
963       \and \(\@DT@loopN>\c@padzeroesN\)}%
964     {\@Hexadecimal\@DT@modctr}%
965     \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
966     \multiply\@DT@modctr by \@strctr
967     \advance\@DT@X by -\@DT@modctr
968     \divide\@strctr by 16\relax
969     \advance\@DT@loopN by -1\relax
970   \ifnum\@strctr>1
971     \repeat
972     \@Hexadecimal\@DT@X
973 }
974
975 \let\Hexadecimalnum=\@Hexadecimal
```

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

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

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

```

995 \newcommand*{\@AAAlph}[1]{%
996   \DT@loopN=#1\relax
997   \advance\DT@loopN by -1\relax
998   \divide\DT@loopN by 26\relax
999   \DT@modctr=\DT@loopN
1000  \multiply\DT@modctr by 26\relax
1001  \DT@X=#1\relax
1002  \advance\DT@X by -1\relax
1003  \advance\DT@X by -\DT@modctr
1004  \advance\DT@loopN by 1\relax
1005  \advance\DT@X by 1\relax
1006  \loop
1007    \@Alph\DT@X
1008    \advance\DT@loopN by -1\relax
1009    \ifnum\DT@loopN>0
1010    \repeat
1011 }
1012
1013 \let\AAAlphnum=\@AAAlph

```

\abalphnum Lowercase alphabetical representation

```

1014 \newcommand*{\@abalph}[1]{%
1015   \ifnum#1>17576\relax
1016     \PackageError{fmtcount}%
1017     {Value of counter too large for \protect\@abalph}%
1018     {Maximum value 17576}%
1019   \else
1020     \DT@padzeroestrue
1021     \@strctr=17576\relax
1022     \DT@X=#1\relax
1023     \advance\DT@X by -1\relax
1024     \loop
1025       \DT@modctr=\DT@X
1026       \divide\DT@modctr by \@strctr
1027       \ifthenelse{\boolean{\DT@padzeroes}
1028         \and \(\DT@modctr=1\)}%
1029       {\@alph\DT@modctr}%
1030       \ifnum\DT@modctr=1\else\DT@padzeroesfalse\fi
1031       \multiply\DT@modctr by \@strctr
1032       \advance\DT@X by -\DT@modctr
1033       \divide\@strctr by 26\relax
1034     \ifnum\@strctr>1
1035     \repeat
1036     \advance\DT@X by 1\relax
1037     \@alph\DT@X
1038   \fi
1039 }
1040
1041 \let\abalphnum=\@abalph

```

`\ABAlphnum` Uppercase alphabetical representation

```

1042 \newcommand*{\@ABAlph}[1]{%
1043   \ifnum#1>17576\relax
1044     \PackageError{fmtcount}%
1045       {Value of counter too large for \protect\@ABAlph}%
1046       {Maximum value 17576}%
1047   \else
1048     \@DT@padzeroestrue
1049     \@strctr=17576\relax
1050     \@DT@X=#1\relax
1051     \advance\@DT@X by -1\relax
1052     \loop
1053       \@DT@modctr=\@DT@X
1054       \divide\@DT@modctr by \@strctr
1055       \ifthenelse{\boolean{\@DT@padzeroes}\and
1056         {(\@DT@modctr=1)}}{\@Alph\@DT@modctr}%
1057       \ifnum\@DT@modctr=1\else\@DT@padzeroesfalse\fi
1058       \multiply\@DT@modctr by \@strctr
1059       \advance\@DT@X by -\@DT@modctr
1060       \divide\@strctr by 26\relax
1061     \ifnum\@strctr>1
1062     \repeat
1063     \advance\@DT@X by 1\relax
1064     \@Alph\@DT@X
1065   \fi
1066 }
1067
1068 \let\ABAlphnum=\@ABAlph

```

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

```

1069 \def\@fmtc@count#1#2\relax{%
1070   \if\relax#1%
1071   \else
1072     \advance\@strctr by 1\relax
1073     \@fmtc@count#2\relax
1074   \fi
1075 }

```

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

```

1076 \newcommand{\@decimal}[1]{%
1077   \@strctr=0\relax
1078   \expandafter\@fmtc@count\number#1\relax
1079   \@DT@loopN=\c@padzeroesN
1080   \advance\@DT@loopN by -\@strctr
1081   \ifnum\@DT@loopN>0\relax
1082     \@strctr=0\relax
1083     \whiledo{\@strctr < \@DT@loopN}{0\advance\@strctr by 1\relax}%
1084   \fi

```

```

1085 \number#1\relax
1086 }
1087
1088 \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.

```

1089 \newcommand{\FCordinal}[1]{%
1090   \expandafter\protect\expandafter\ordinalnum{%
1091     \expandafter\the\csname c@#1\endcsname}%
1092 }

```

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

```

1093 \@ifundefined{ordinal}
1094 {\let\ordinal\FCordinal}%
1095 {%
1096   \PackageWarning{fmtcount}%
1097   {\string\ordinal \space already defined use
1098    \string\FCordinal \space instead.}
1099 }

```

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

```

1100 \newcommand*{\ordinalnum}[1]{%
1101   \new@ifnextchar[%
1102     {\@ordinalnum{#1}}%
1103     {\@ordinalnum{#1}[m]}%
1104 }

```

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

```

1105 \def\@ordinalnum#1[#2]{%
1106   {%

```

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

```

1107 \ifthenelse{\equal{#2}{f}}%
1108 {%
1109 \protect\@ordinalF{#1}{\@fc@ordstr}%
1110 }%
1111 {%
1112 \ifthenelse{\equal{#2}{n}}%
1113 {%
1114 \protect\@ordinalN{#1}{\@fc@ordstr}%
1115 }%
1116 {%
1117 \ifthenelse{\equal{#2}{m}}%
1118 {}%
1119 {%
1120 \PackageError{fmtcount}%
1121 {Invalid gender option ‘#2’}%
1122 {Available options are m, f or n}%
1123 }%
1124 \protect\@ordinalM{#1}{\@fc@ordstr}%
1125 }%
1126 }%
1127 \@fc@ordstr
1128 }%
1129 }

```

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

```

1130 \newcommand*{\storeordinal}[2]{%
1131 \expandafter\protect\expandafter\storeordinalnum{#1}{%
1132 \expandafter\the\csname c@#2\endcsname}%
1133 }

```

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

```

1134 \newcommand*{\storeordinalnum}[2]{%
1135 \@ifnextchar[%
1136 {\@storeordinalnum{#1}{#2}}%
1137 {\@storeordinalnum{#1}{#2}[m]}%
1138 }

```

`\@storeordinalnum` Store ordinal according to gender:

```

1139 \def\@storeordinalnum#1#2[#3]{%
1140 \ifthenelse{\equal{#3}{f}}%
1141 {%
1142 \protect\@ordinalF{#2}{\@fc@ord}
1143 }%
1144 {%
1145 \ifthenelse{\equal{#3}{n}}%
1146 {%
1147 \protect\@ordinalN{#2}{\@fc@ord}%

```

```

1148 }%
1149 {%
1150 \ifthenelse{\equal{#3}{m}}%
1151 {}%
1152 {%
1153 \PackageError{fmtcount}%
1154 {Invalid gender option ‘#3’}%
1155 {Available options are m or f}%
1156 }%
1157 \protect\@ordinalM{#2}{\@fc@ord}%
1158 }%
1159 }%
1160 \expandafter\let\csname @fcs@#1\endcsname\@fc@ord
1161 }

```

\FMCuse Get stored information:

```

1162 \newcommand*\@FMCuse{1}{\csname @fcs@#1\endcsname}

```

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

```

1163 \newcommand*\@ordinalstring{1}{%
1164 \expandafter\protect\expandafter\ordinalstringnum{%
1165 \expandafter\the\csname c@#1\endcsname}%
1166 }

```

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

```

1167 \newcommand*\@ordinalstringnum{1}{%
1168 \new@ifnextchar[%
1169 {\@ordinal@string{#1}}%
1170 {\@ordinal@string{#1}[m]}%
1171 }

```

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

```

1172 \def\@ordinal@string#1[#2]{%
1173 {%
1174 \ifthenelse{\equal{#2}{f}}%
1175 {%
1176 \protect\@ordinalstringF{#1}{\@fc@ordstr}%
1177 }%
1178 {%
1179 \ifthenelse{\equal{#2}{n}}%
1180 {%
1181 \protect\@ordinalstringN{#1}{\@fc@ordstr}%
1182 }%
1183 {%
1184 \ifthenelse{\equal{#2}{m}}%
1185 {}%
1186 {%
1187 \PackageError{fmtcount}%
1188 {Invalid gender option ‘#2’ to \string\ordinalstring}%

```



```

1189         {Available options are m, f or f}%
1190     }%
1191     \protect\@ordinalstringM{#1}{\@fc@ordstr}%
1192 }%
1193 }%
1194 \@fc@ordstr
1195 }%
1196 }

```

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

```

1197 \newcommand*\storeordinalstring}[2]{%
1198   \expandafter\protect\expandafter\storeordinalstringnum{#1}{%
1199     \expandafter\the\csname c@#2\endcsname}%
1200 }

```

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

```

1201 \newcommand*\storeordinalstringnum}[2]{%
1202   \@ifnextchar[%
1203     {\@store@ordinal@string{#1}{#2}}%
1204     {\@store@ordinal@string{#1}{#2}[m]}%
1205 }

```

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

```

1206 \def\@store@ordinal@string#1#2[#3]{%
1207   \ifthenelse{\equal{#3}{f}}{%
1208     {%
1209       \protect\@ordinalstringF{#2}{\@fc@ordstr}%
1210     }%
1211     {%
1212       \ifthenelse{\equal{#3}{n}}{%
1213         {%
1214           \protect\@ordinalstringN{#2}{\@fc@ordstr}%
1215         }%
1216         {%
1217           \ifthenelse{\equal{#3}{m}}{%
1218             {}%
1219           }%
1220           \PackageError{fmtcount}%
1221             {Invalid gender option ‘#3’ to \string\ordinalstring}%
1222             {Available options are m, f or n}%
1223           }%
1224           \protect\@ordinalstringM{#2}{\@fc@ordstr}%
1225         }%
1226       }%
1227       \expandafter\let\csname @fcs@#1\endcsname\@fc@ordstr
1228 }

```

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

```
1229 \newcommand*{\Ordinalstring}[1]{%
1230   \expandafter\protect\expandafter\Ordinalstringnum{%
1231     \expandafter\the\csname c@#1\endcsname}%
1232 }
```

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

```
1233 \newcommand*{\Ordinalstringnum}[1]{%
1234   \new@ifnextchar[%
1235     {\@Ordinal@string{#1}}%
1236     {\@Ordinal@string{#1}[m]}%
1237 }
```

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

```
1238 \def\@Ordinal@string#1[#2]{%
1239   {%
1240     \ifthenelse{\equal{#2}{f}}%
1241     {%
1242       \protect\@OrdinalstringF{#1}{\@fc@ordstr}%
1243     }%
1244     {%
1245       \ifthenelse{\equal{#2}{n}}%
1246       {%
1247         \protect\@OrdinalstringN{#1}{\@fc@ordstr}%
1248       }%
1249       {%
1250         \ifthenelse{\equal{#2}{m}}%
1251         {%
1252           {%
1253             \PackageError{fmtcount}%
1254               {Invalid gender option ‘#2’}%
1255               {Available options are m, f or n}%
1256           }%
1257           \protect\@OrdinalstringM{#1}{\@fc@ordstr}%
1258         }%
1259       }%
1260     }%
1261   }%
1262 }
```

`\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.

```
1263 \newcommand*{\storeOrdinalstring}[2]{%
1264   \expandafter\protect\expandafter\storeOrdinalstringnum{#1}{%
1265     \expandafter\the\csname c@#2\endcsname}%
1266 }
```

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

```
1267 \newcommand*{\storeOrdinalstringnum}[2]{%
1268   \@ifnextchar[%
1269     {\@store@Ordinal@string{#1}{#2}}%
1270     {\@store@Ordinal@string{#1}{#2}[m]}}%
1271 }
```

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

```
1272 \def\@store@Ordinal@string#1#2[#3]{%
1273   \ifthenelse{\equal{#3}{f}}{%
1274     {%
1275       \protect\@OrdinalstringF{#2}{\@fc@ordstr}%
1276     }%
1277     {%
1278       \ifthenelse{\equal{#3}{n}}{%
1279         {%
1280           \protect\@OrdinalstringN{#2}{\@fc@ordstr}%
1281         }%
1282         {%
1283           \ifthenelse{\equal{#3}{m}}{%
1284             {}%
1285             {%
1286               \PackageError{fmtcount}%
1287                 {Invalid gender option ‘#3’}%
1288                 {Available options are m or f}%
1289             }%
1290           \protect\@OrdinalstringM{#2}{\@fc@ordstr}%
1291         }%
1292       }%
1293     \expandafter\let\csname @fcs@#1\endcsname\@fc@ordstr
1294 }
```

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

```
1295 \newcommand*{\storeORDINALstring}[2]{%
1296   \expandafter\protect\expandafter\storeORDINALstringnum{#1}{%
1297     \expandafter\the\csname c@#2\endcsname}%
1298 }
```

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

```
1299 \newcommand*{\storeORDINALstringnum}[2]{%
1300   \@ifnextchar[%
1301     {\@store@ORDINAL@string{#1}{#2}}%
1302     {\@store@ORDINAL@string{#1}{#2}[m]}}%
1303 }
```

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

```

1304 \def\@store@ORDINAL@string#1#2[#3]{%
1305   \ifthenelse{\equal{#3}{f}}{%
1306     {%
1307       \protect\@ordinalstringF{#2}{\@fc@ordstr}%
1308     }%
1309     {%
1310       \ifthenelse{\equal{#3}{n}}{%
1311         {%
1312           \protect\@ordinalstringN{#2}{\@fc@ordstr}%
1313         }%
1314         {%
1315           \ifthenelse{\equal{#3}{m}}{%
1316             {}%
1317             {%
1318               \PackageError{fmtcount}%
1319                 {Invalid gender option ‘#3’}%
1320                 {Available options are m or f}%
1321             }%
1322           \protect\@ordinalstringM{#2}{\@fc@ordstr}%
1323         }%
1324       }%
1325       \expandafter\edef\csname @fcs@#1\endcsname{%
1326         \noexpand\MakeUppercase{\@fc@ordstr}%
1327       }%
1328 }

```

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

```

1329 \newcommand*\ORDINALstring[1]{%
1330   \expandafter\protect\expandafter\ORDINALstringnum{%
1331     \expandafter\the\csname c@#1\endcsname
1332   }%
1333 }

```

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

```

1334 \newcommand*\ORDINALstringnum[1]{%
1335   \new@ifnextchar[%
1336     {\ORDINALstring{#1}}%
1337     {\ORDINALstring{#1}[m]}%
1338 }

```

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

```

1339 \def\@ORDINAL@string#1[#2]{%
1340   {%
1341     \ifthenelse{\equal{#2}{f}}{%
1342       {%
1343         \protect\@ordinalstringF{#1}{\@fc@ordstr}%
1344       }%

```

```

1345   {%
1346     \ifthenelse{\equal{#2}{n}}{%
1347       {%
1348         \protect\@ordinalstringN{#1}{\@fc@ordstr}%
1349       }%
1350     {%
1351       \ifthenelse{\equal{#2}{m}}{%
1352         {}%
1353       {%
1354         \PackageError{fmtcount}%
1355           {Invalid gender option ‘#2’}%
1356           {Available options are m, f or n}%
1357       }%
1358       \protect\@ordinalstringM{#1}{\@fc@ordstr}%
1359     }%
1360   }%
1361   \MakeUppercase{\@fc@ordstr}%
1362 }%
1363 }

```

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

```

1364 \newcommand*{\storenumberstring}[2]{%
1365   \expandafter\protect\expandafter\storenumberstringnum{#1}{%
1366     \expandafter\the\csname c@#2\endcsname}%
1367 }

```

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

```

1368 \newcommand{\storenumberstringnum}[2]{%
1369   \@ifnextchar[%
1370     {\@store@number@string{#1}{#2}}%
1371     {\@store@number@string{#1}{#2}[m]}%
1372 }

```

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

```

1373 \def\@store@number@string#1#2[#3]{%
1374   \ifthenelse{\equal{#3}{f}}{%
1375     {%
1376       \protect\@numberstringF{#2}{\@fc@numstr}%
1377     }%
1378   {%
1379     \ifthenelse{\equal{#3}{n}}{%
1380       {%
1381         \protect\@numberstringN{#2}{\@fc@numstr}%
1382       }%
1383     {%
1384       \ifthenelse{\equal{#3}{m}}{%
1385         {}%
1386       {%

```

```

1387     \PackageError{fmtcount}
1388     {Invalid gender option ‘#3’}%
1389     {Available options are m, f or n}%
1390   }%
1391   \protect\@numberstringM{#2}{\@fc@numstr}%
1392 }%
1393 }%
1394 \expandafter\let\csname @fcs@#1\endcsname\@fc@numstr
1395 }

```

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

```

1396 \newcommand*{\numberstring}[1]{%
1397   \expandafter\protect\expandafter\numberstringnum{%
1398     \expandafter\the\csname c@#1\endcsname}%
1399 }

```

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

```

1400 \newcommand*{\numberstringnum}[1]{%
1401   \new@ifnextchar[%
1402     {\@number@string{#1}}%
1403     {\@number@string{#1}[m]}%
1404 }

```

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

```

1405 \def\@number@string#1[#2]{%
1406   {%
1407     \ifthenelse{\equal{#2}{f}}%
1408     {%
1409       \protect\@numberstringF{#1}{\@fc@numstr}%
1410     }%
1411     {%
1412       \ifthenelse{\equal{#2}{n}}%
1413       {%
1414         \protect\@numberstringN{#1}{\@fc@numstr}%
1415       }%
1416       {%
1417         \ifthenelse{\equal{#2}{m}}%
1418         {}%
1419         {%
1420           \PackageError{fmtcount}%
1421           {Invalid gender option ‘#2’}%
1422           {Available options are m, f or n}%
1423         }%
1424         \protect\@numberstringM{#1}{\@fc@numstr}%
1425       }%
1426     }%
1427     \@fc@numstr
1428   }%
1429 }

```

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

```
1430 \newcommand*\storeNumberstring}[2]{%
1431   \expandafter\protect\expandafter\storeNumberstringnum{#1}{%
1432     \expandafter\the\csname c@#2\endcsname}%
1433 }
```

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

```
1434 \newcommand*\storeNumberstringnum}[2]{%
1435   \@ifnextchar[%
1436     {\@store@Number@string{#1}{#2}}%
1437     {\@store@Number@string{#1}{#2}[m]}%
1438 }
```

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

```
1439 \def\@store@Number@string#1#2[#3]{%
1440   \ifthenelse{\equal{#3}{f}}{%
1441     {%
1442       \protect\@NumberstringF{#2}{\@fc@numstr}%
1443     }%
1444     {%
1445       \ifthenelse{\equal{#3}{n}}{%
1446         {%
1447           \protect\@NumberstringN{#2}{\@fc@numstr}%
1448         }%
1449         {%
1450           \ifthenelse{\equal{#3}{m}}{%
1451             {}%
1452             {%
1453               \PackageError{fmtcount}%
1454                 {Invalid gender option ‘#3’}%
1455                 {Available options are m, f or n}%
1456             }%
1457           \protect\@NumberstringM{#2}{\@fc@numstr}%
1458         }%
1459       }%
1460       \expandafter\let\csname @fcs@#1\endcsname\@fc@numstr
1461 }
```

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

```
1462 \newcommand*\Numberstring}[1]{%
1463   \expandafter\protect\expandafter\Numberstringnum{%
1464     \expandafter\the\csname c@#1\endcsname}%
1465 }
```

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

```
1466 \newcommand*\Numberstringnum}[1]{%
1467   \new@ifnextchar[%
```

```

1468 {\@Number@string{#1}}%
1469 {\@Number@string{#1}[m]}%
1470 }

```

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

```

1471 \def\@Number@string#1[#2]{%
1472   {%
1473     \ifthenelse{\equal{#2}{f}}%
1474     {%
1475       \protect\@NumberstringF{#1}{\@fc@numstr}%
1476     }%
1477     {%
1478       \ifthenelse{\equal{#2}{n}}%
1479       {%
1480         \protect\@NumberstringN{#1}{\@fc@numstr}%
1481       }%
1482       {%
1483         \ifthenelse{\equal{#2}{m}}%
1484         {}%
1485         {%
1486           \PackageError{fmtcount}%
1487             {Invalid gender option ‘#2’}%
1488             {Available options are m, f or n}%
1489         }%
1490         \protect\@NumberstringM{#1}{\@fc@numstr}%
1491       }%
1492     }%
1493     \@fc@numstr
1494   }%
1495 }

```

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

```

1496 \newcommand{\storeNUMBERstring}[2]{%
1497   \expandafter\protect\expandafter\storeNUMBERstringnum{#1}{%
1498     \expandafter\the\csname c@#2\endcsname}%
1499 }

```

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

```

1500 \newcommand{\storeNUMBERstringnum}[2]{%
1501   \@ifnextchar[%
1502   {\@store@NUMBER@string{#1}{#2}}%
1503   {\@store@NUMBER@string{#1}{#2}[m]}%
1504 }

```

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

```

1505 \def\@store@NUMBER@string#1#2[#3]{%
1506   \ifthenelse{\equal{#3}{f}}%
1507   {%

```



```

1508 \protect\@numberstringF{#2}{\@fc@numstr}%
1509 }%
1510 {%
1511 \ifthenelse{\equal{#3}{n}}%
1512 {%
1513 \protect\@numberstringN{#2}{\@fc@numstr}%
1514 }%
1515 {%
1516 \ifthenelse{\equal{#3}{m}}%
1517 {}%
1518 {%
1519 \PackageError{fmtcount}%
1520 {Invalid gender option ‘#3’}%
1521 {Available options are m or f}%
1522 }%
1523 \protect\@numberstringM{#2}{\@fc@numstr}%
1524 }%
1525 }%
1526 \expandafter\edef\csname @fcs@#1\endcsname{%
1527 \noexpand\MakeUppercase{\@fc@numstr}%
1528 }%
1529 }

```

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

```

1530 \newcommand*{\NUMBERstring}[1]{%
1531 \expandafter\protect\expandafter\NUMBERstringnum{%
1532 \expandafter\the\csname c@#1\endcsname}%
1533 }

```

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

```

1534 \newcommand*{\NUMBERstringnum}[1]{%
1535 \new@ifnextchar[%
1536 {\@NUMBER@string{#1}}%
1537 {\@NUMBER@string{#1}[m]}%
1538 }

```

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

```

1539 \def\@NUMBER@string#1[#2]{%
1540 {%
1541 \ifthenelse{\equal{#2}{f}}%
1542 {%
1543 \protect\@numberstringF{#1}{\@fc@numstr}%
1544 }%
1545 {%
1546 \ifthenelse{\equal{#2}{n}}%
1547 {%
1548 \protect\@numberstringN{#1}{\@fc@numstr}%
1549 }%

```

```

1550      {%
1551      \ifthenelse{\equal{#2}{m}}{%
1552      }{%
1553      {%
1554      \PackageError{fmtcount}%
1555      {Invalid gender option ‘#2’}%
1556      {Available options are m, f or n}%
1557      }%
1558      \protect\@numberstringM{#1}{\@fc@numstr}%
1559      }%
1560      }%
1561      \MakeUppercase{\@fc@numstr}%
1562      }%
1563 }

```

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

```

1564 \providecommand*\binary}[1]{%
1565 \expandafter\protect\expandafter\@binary{%
1566 \expandafter\the\csname c@#1\endcsname}%
1567 }

```

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

```

1568 \providecommand*\aaalph}[1]{%
1569 \expandafter\protect\expandafter\@aaalph{%
1570 \expandafter\the\csname c@#1\endcsname}%
1571 }

```

`\AAAAlph` As before, but upper case.

```

1572 \providecommand*\AAAAlph}[1]{%
1573 \expandafter\protect\expandafter\@AAAAlph{%
1574 \expandafter\the\csname c@#1\endcsname}%
1575 }

```

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

```

1576 \providecommand*\abalph}[1]{%
1577 \expandafter\protect\expandafter\@abalph{%
1578 \expandafter\the\csname c@#1\endcsname}%
1579 }

```

`\ABAlph` As above, but upper case.

```

1580 \providecommand*\ABAlph}[1]{%
1581 \expandafter\protect\expandafter\@ABAlph{%
1582 \expandafter\the\csname c@#1\endcsname}%
1583 }

```

`\hexadecimal` Hexadecimal:

```

1584 \providecommand*\hexadecimal}[1]{%
1585 \expandafter\protect\expandafter\@hexadecimal{%
1586 \expandafter\the\csname c@#1\endcsname}%
1587 }

```

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

```
1588 \providecommand*{\Hexadecimal}[1]{%
1589   \expandafter\protect\expandafter\@Hexadecimal{%
1590     \expandafter\the\csname c@#1\endcsname}%
1591 }
```

`\octal` Octal:

```
1592 \providecommand*{\octal}[1]{%
1593   \expandafter\protect\expandafter\@octal{%
1594     \expandafter\the\csname c@#1\endcsname}%
1595 }
```

`\decimal` Decimal:

```
1596 \providecommand*{\decimal}[1]{%
1597   \expandafter\protect\expandafter\@decimal{%
1598     \expandafter\the\csname c@#1\endcsname}%
1599 }
```

9.4 Multilingual Definitions

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

```
1600 \def\@setdef@ultfmtcount{%
1601   \@ifundefined{@OrdinalMenglish}{\FCloadlang{english}}{}%
1602   \def\@OrdinalstringM{\@OrdinalstringMenglish}%
1603   \let\@OrdinalstringF=\@OrdinalstringMenglish
1604   \let\@OrdinalstringN=\@OrdinalstringMenglish
1605   \def\@OrdinalstringM{\@OrdinalstringMenglish}%
1606   \let\@OrdinalstringF=\@OrdinalstringMenglish
1607   \let\@OrdinalstringN=\@OrdinalstringMenglish
1608   \def\@NumberstringM{\@NumberstringMenglish}%
1609   \let\@NumberstringF=\@NumberstringMenglish
1610   \let\@NumberstringN=\@NumberstringMenglish
1611   \def\@NumberstringM{\@NumberstringMenglish}%
1612   \let\@NumberstringF=\@NumberstringMenglish
1613   \let\@NumberstringN=\@NumberstringMenglish
1614   \def\@OrdinalM{\@OrdinalMenglish}%
1615   \let\@OrdinalF=\@OrdinalM
1616   \let\@OrdinalN=\@OrdinalM
1617 }
```

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

```
1618 \newcommand*{\fc@multiling}[2]{%
1619   \ifcsundef{#1#2\languagename}%
1620   {% try loading it
1621     \FCloadlang{\languagename}%
1622   }%
```

```

1623 {%
1624 }%
1625 \ifcsundef{@#1#2\language}%
1626 {%
1627   \PackageWarning{fmtcount}%
1628   {No support for \expandafter\string\csname#1\endcsname\space for
1629    language '\language}%
1630   \ifthenelse{\equal{\language}{\fc@mainlang}}%
1631   {%
1632     \FCloadlang{english}%
1633   }%
1634   {%
1635   }%
1636   \ifcsdef{@#1#2\fc@mainlang}%
1637   {%
1638     \csuse{@#1#2\fc@mainlang}%
1639   }%
1640   {%
1641     \PackageWarningNoLine{fmtcount}%
1642     {No languages loaded at all! Loading english definitions}%
1643     \FCloadlang{english}%
1644     \def\fc@mainlang{english}%
1645     \csuse{@#1#2english}%
1646   }%
1647 }%
1648 {%
1649   \csuse{@#1#2\language}%
1650 }%
1651 }

```

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

```

1652 \def\@set@multiling@fmtcount{%
    The masculine version of \numberstring:
1653   \def\@numberstringM{%
1654     \fc@multiling{numberstring}{M}%
1655   }%
    The feminine version of \numberstring:
1656   \def\@numberstringF{%
1657     \fc@multiling{numberstring}{F}%
1658   }%
    The neuter version of \numberstring:
1659   \def\@numberstringN{%
1660     \fc@multiling{numberstring}{N}%
1661   }%
    The masculine version of \Numberstring:
1662   \def\@NumberstringM{%
1663     \fc@multiling{Numberstring}{M}%

```

```

1664 }%

The feminine version of \Numberstring:
1665 \def\@NumberstringF{%
1666     \fc@multiling{Numberstring}{F}%
1667 }%

The neuter version of \Numberstring:
1668 \def\@NumberstringN{%
1669     \fc@multiling{Numberstring}{N}%
1670 }%

The masculine version of \ordinal:
1671 \def\@ordinalM{%
1672     \fc@multiling{ordinal}{M}%
1673 }%

The feminine version of \ordinal:
1674 \def\@ordinalF{%
1675     \fc@multiling{ordinal}{F}%
1676 }%

The neuter version of \ordinal:
1677 \def\@ordinalN{%
1678     \fc@multiling{ordinal}{N}%
1679 }%

The masculine version of \ordinalstring:
1680 \def\@ordinalstringM{%
1681     \fc@multiling{ordinalstring}{M}%
1682 }%

The feminine version of \ordinalstring:
1683 \def\@ordinalstringF{%
1684     \fc@multiling{ordinalstring}{F}%
1685 }%

The neuter version of \ordinalstring:
1686 \def\@ordinalstringN{%
1687     \fc@multiling{ordinalstring}{N}%
1688 }%

The masculine version of \Ordinalstring:
1689 \def\@OrdinalstringM{%
1690     \fc@multiling{Ordinalstring}{M}%
1691 }%

The feminine version of \Ordinalstring:
1692 \def\@OrdinalstringF{%
1693     \fc@multiling{Ordinalstring}{F}%
1694 }%

```

The neuter version of \Ordinalstring:

```
1695 \def\@OrdinalstringN{%
1696   \fc@multiling{Ordinalstring}{N}%
1697 }%
1698 }
```

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

```
1699 \@ifpackageloaded{babel}%
1700 {%
1701   \@set@multiling@fmtcount
1702 }%
1703 {%
1704   \@ifpackageloaded{ngerman}%
1705   {%
1706     \FCloadlang{ngerman}%
1707     \@set@multiling@fmtcount
1708   }%
1709   {%
1710     \@setdef@ultfmtcount
1711   }%
1712 }
```

Backwards compatibility:

```
1713 \let\@ordinal=\@ordinalM
1714 \let\@ordinalstring=\@ordinalstringM
1715 \let\@Ordinalstring=\@OrdinalstringM
1716 \let\@numberstring=\@numberstringM
1717 \let\@Numberstring=\@NumberstringM
```

9.4.1 fc-american.def

American English definitions

```
1718 \ProvidesFCLanguage{american}[2012/06/18]
```

Loaded fc-USenglish.def if not already loaded

```
1719 \FCloadlang{USenglish}
```

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

```
1720 \let\@ordinalMamerican\@ordinalMUSenglish
1721 \let\@ordinalFamerican\@ordinalMUSenglish
1722 \let\@ordinalNamerican\@ordinalMUSenglish
1723 \let\@numberstringMamerican\@numberstringMUSenglish
1724 \let\@numberstringFamerican\@numberstringMUSenglish
1725 \let\@numberstringNamerican\@numberstringMUSenglish
1726 \let\@NumberstringMamerican\@NumberstringMUSenglish
1727 \let\@NumberstringFamerican\@NumberstringMUSenglish
1728 \let\@NumberstringNamerican\@NumberstringMUSenglish
1729 \let\@ordinalstringMamerican\@ordinalstringMUSenglish
1730 \let\@ordinalstringFamerican\@ordinalstringMUSenglish
1731 \let\@ordinalstringNamerican\@ordinalstringMUSenglish
```

```

1732 \let\@OrdinalstringMamerican\@OrdinalstringMUSenglish
1733 \let\@OrdinalstringFamerican\@OrdinalstringMUSenglish
1734 \let\@OrdinalstringNamerican\@OrdinalstringMUSenglish

```

9.4.2 fc-british.def

British definitions

```

1735 \ProvidesFCLanguage{british}[2012/06/18]

```

Load fc-english.def, if not already loaded

```

1736 \FCloadlang{english}

```

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

```

1737 \let\@ordinalMbritish\@ordinalMenglish
1738 \let\@ordinalFbritish\@ordinalMenglish
1739 \let\@ordinalNbritish\@ordinalMenglish
1740 \let\@numberstringMbritish\@numberstringMenglish
1741 \let\@numberstringFbritish\@numberstringMenglish
1742 \let\@numberstringNbritish\@numberstringMenglish
1743 \let\@NumberstringMbritish\@NumberstringMenglish
1744 \let\@NumberstringFbritish\@NumberstringMenglish
1745 \let\@NumberstringNbritish\@NumberstringMenglish
1746 \let\@ordinalstringMbritish\@ordinalstringMenglish
1747 \let\@ordinalstringFbritish\@ordinalstringMenglish
1748 \let\@ordinalstringNbritish\@ordinalstringMenglish
1749 \let\@OrdinalstringMbritish\@OrdinalstringMenglish
1750 \let\@OrdinalstringFbritish\@OrdinalstringMenglish
1751 \let\@OrdinalstringNbritish\@OrdinalstringMenglish

```

9.4.3 fc-english.def

English definitions

```

1752 \ProvidesFCLanguage{english}[2012/06/18]

```

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

```

1753 \newcommand*{\@ordinalMenglish}[2]{%
1754 \def\@fc@ord{}%
1755 \@orgargctr=#1\relax
1756 \@ordinalctr=#1%
1757 \@modulo{\@ordinalctr}{100}%
1758 \ifnum\@ordinalctr=11\relax
1759   \def\@fc@ord{th}%
1760 \else
1761   \ifnum\@ordinalctr=12\relax
1762     \def\@fc@ord{th}%
1763   \else
1764     \ifnum\@ordinalctr=13\relax

```

```

1765     \def\@fc@ord{th}%
1766   \else
1767     \@modulo{\@ordinalctr}{10}%
1768     \ifcase\@ordinalctr
1769       \def\@fc@ord{th}%      case 0
1770       \or \def\@fc@ord{st}%  case 1
1771       \or \def\@fc@ord{nd}%  case 2
1772       \or \def\@fc@ord{rd}%  case 3
1773     \else
1774       \def\@fc@ord{th}%      default case
1775     \fi
1776   \fi
1777 \fi
1778 \fi
1779 \edef#2{\number#1\relax\noexpand\fmtord{\@fc@ord}}%
1780 }

```

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

```

1781 \let\@ordinalFenglish=\@ordinalMenglish
1782 \let\@ordinalNenglish=\@ordinalMenglish

```

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

```

1783 \newcommand*\@@unitstringenglish}[1]{%
1784 \ifcase#1\relax
1785 zero%
1786 \or one%
1787 \or two%
1788 \or three%
1789 \or four%
1790 \or five%
1791 \or six%
1792 \or seven%
1793 \or eight%
1794 \or nine%
1795 \fi
1796 }

```

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

```

1797 \newcommand*\@@tenstringenglish}[1]{%
1798 \ifcase#1\relax
1799 \or ten%
1800 \or twenty%
1801 \or thirty%
1802 \or forty%
1803 \or fifty%
1804 \or sixty%
1805 \or seventy%

```



```

1806 \or eighty%
1807 \or ninety%
1808 \fi
1809 }

```

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

```

1810 \newcommand*\@@teenstringenglish}[1]{%
1811 \ifcase#1\relax
1812 ten%
1813 \or eleven%
1814 \or twelve%
1815 \or thirteen%
1816 \or fourteen%
1817 \or fifteen%
1818 \or sixteen%
1819 \or seventeen%
1820 \or eighteen%
1821 \or nineteen%
1822 \fi
1823 }

```

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

```

1824 \newcommand*\@@Unitstringenglish}[1]{%
1825 \ifcase#1\relax
1826 Zero%
1827 \or One%
1828 \or Two%
1829 \or Three%
1830 \or Four%
1831 \or Five%
1832 \or Six%
1833 \or Seven%
1834 \or Eight%
1835 \or Nine%
1836 \fi
1837 }

```

The tens:

```

1838 \newcommand*\@@Tenstringenglish}[1]{%
1839 \ifcase#1\relax
1840 \or Ten%
1841 \or Twenty%
1842 \or Thirty%
1843 \or Forty%
1844 \or Fifty%
1845 \or Sixty%
1846 \or Seventy%
1847 \or Eighty%
1848 \or Ninety%
1849 \fi
1850 }

```

The teens:

```
1851 \newcommand*{\@@Teenstringenglish}[1]{%
1852 \ifcase#1\relax
1853 Ten%
1854 \or Eleven%
1855 \or Twelve%
1856 \or Thirteen%
1857 \or Fourteen%
1858 \or Fifteen%
1859 \or Sixteen%
1860 \or Seventeen%
1861 \or Eighteen%
1862 \or Nineteen%
1863 \fi
1864 }
```

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.)

```
1865 \newcommand*{\@@numberstringenglish}[2]{%
1866 \ifnum#1>999999
1867 \PackageError{fmtcount}{Out of range}%
1868 {This macro only works for values less than 100000}%
1869 \else
1870 \ifnum#1<0
1871 \PackageError{fmtcount}{Negative numbers not permitted}%
1872 {This macro does not work for negative numbers, however
1873 you can try typing "minus" first, and then pass the modulus of
1874 this number}%
1875 \fi
1876 \fi
1877 \def#2{}%
1878 \@strctr=#1\relax \divide\@strctr by 1000\relax
1879 \ifnum\@strctr>9
1880 \divide\@strctr by 10
1881 \ifnum\@strctr>1\relax
1882 \let\@@fc@numstr#2\relax
1883 \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
1884 \@strctr=#1 \divide\@strctr by 1000\relax
1885 \@modulo{\@strctr}{10}%
1886 \ifnum\@strctr>0\relax
1887 \let\@@fc@numstr#2\relax
1888 \edef#2{\@@fc@numstr-\@unitstring{\@strctr}}%
1889 \fi
1890 \else
1891 \@strctr=#1\relax
1892 \divide\@strctr by 1000\relax
1893 \@modulo{\@strctr}{10}%

```

```

1894 \let\@@fc@numstr#2\relax
1895 \edef#2{\@@fc@numstr\@teenstring{\@strctr}}}%
1896 \fi
1897 \let\@@fc@numstr#2\relax
1898 \edef#2{\@@fc@numstr\ \@thousand}%
1899 \else
1900 \ifnum\@strctr>0\relax
1901 \let\@@fc@numstr#2\relax
1902 \edef#2{\@@fc@numstr\@unitstring{\@strctr}\ \@thousand}%
1903 \fi
1904 \fi
1905 \@strctr=#1\relax \@modulo{\@strctr}{1000}%
1906 \divide\@strctr by 100
1907 \ifnum\@strctr>0\relax
1908 \ifnum#1>1000\relax
1909 \let\@@fc@numstr#2\relax
1910 \edef#2{\@@fc@numstr\ }%
1911 \fi
1912 \let\@@fc@numstr#2\relax
1913 \edef#2{\@@fc@numstr\@unitstring{\@strctr}\ \@hundred}%
1914 \fi
1915 \@strctr=#1\relax \@modulo{\@strctr}{100}%
1916 \ifnum#1>100\relax
1917 \ifnum\@strctr>0\relax
1918 \let\@@fc@numstr#2\relax
1919 \edef#2{\@@fc@numstr\ \@andname\ }%
1920 \fi
1921 \fi
1922 \ifnum\@strctr>19\relax
1923 \divide\@strctr by 10\relax
1924 \let\@@fc@numstr#2\relax
1925 \edef#2{\@@fc@numstr\@tenstring{\@strctr}}}%
1926 \@strctr=#1\relax \@modulo{\@strctr}{10}%
1927 \ifnum\@strctr>0\relax
1928 \let\@@fc@numstr#2\relax
1929 \edef#2{\@@fc@numstr-\@unitstring{\@strctr}}}%
1930 \fi
1931 \else
1932 \ifnum\@strctr<10\relax
1933 \ifnum\@strctr=0\relax
1934 \ifnum#1<100\relax
1935 \let\@@fc@numstr#2\relax
1936 \edef#2{\@@fc@numstr\@unitstring{\@strctr}}}%
1937 \fi
1938 \else
1939 \let\@@fc@numstr#2\relax
1940 \edef#2{\@@fc@numstr\@unitstring{\@strctr}}}%
1941 \fi
1942 \else

```

```

1943 \@modulo{\@strctr}{10}%
1944 \let\@fc@numstr#2\relax
1945 \edef#2{\@fc@numstr\@teenstring{\@strctr}}%
1946 \fi
1947 \fi
1948 }

```

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

```

1949 \DeclareRobustCommand{\@numberstringMenglish}[2]{%
1950 \let\@unitstring=\@unitstringenglish
1951 \let\@teenstring=\@teenstringenglish
1952 \let\@tenstring=\@tenstringenglish
1953 \def\@hundred{hundred}\def\@thousand{thousand}%
1954 \def\@andname{and}%
1955 \@numberstringenglish{#1}{#2}%
1956 }

```

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

```

1957 \let\@numberstringFenglish=\@numberstringMenglish
1958 \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.

```

1959 \newcommand*{\@NumberstringMenglish}[2]{%
1960 \let\@unitstring=\@Unitstringenglish
1961 \let\@teenstring=\@Teenstringenglish
1962 \let\@tenstring=\@Tenstringenglish
1963 \def\@hundred{Hundred}\def\@thousand{Thousand}%
1964 \def\@andname{and}%
1965 \@numberstringenglish{#1}{#2}}

```

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

```

1966 \let\@NumberstringFenglish=\@NumberstringMenglish
1967 \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:

```

1968 \newcommand*{\@unitthstringenglish}[1]{%
1969 \ifcase#1\relax
1970 zeroth%
1971 \or first%
1972 \or second%
1973 \or third%
1974 \or fourth%
1975 \or fifth%
1976 \or sixth%
1977 \or seventh%
1978 \or eighth%
1979 \or ninth%

```

1980 \fi

1981 }

Next the tens:

1982 \newcommand*{\@@tenthstringenglish}[1]{%

1983 \ifcase#1\relax

1984 \or tenth%

1985 \or twentieth%

1986 \or thirtieth%

1987 \or fortieth%

1988 \or fiftieth%

1989 \or sixtieth%

1990 \or seventieth%

1991 \or eightieth%

1992 \or ninetieth%

1993 \fi

1994 }

The teens:

1995 \newcommand*{\@@teenthstringenglish}[1]{%

1996 \ifcase#1\relax

1997 tenth%

1998 \or eleventh%

1999 \or twelfth%

2000 \or thirteenth%

2001 \or fourteenth%

2002 \or fifteenth%

2003 \or sixteenth%

2004 \or seventeenth%

2005 \or eighteenth%

2006 \or nineteenth%

2007 \fi

2008 }

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

2009 \newcommand*{\@@Unitthstringenglish}[1]{%

2010 \ifcase#1\relax

2011 Zeroth%

2012 \or First%

2013 \or Second%

2014 \or Third%

2015 \or Fourth%

2016 \or Fifth%

2017 \or Sixth%

2018 \or Seventh%

2019 \or Eighth%

2020 \or Ninth%

2021 \fi

2022 }

The tens:

```

2023 \newcommand*\@@Tenthstringenglish}[1]{%
2024 \ifcase#1\relax
2025 \or Tenth%
2026 \or Twentieth%
2027 \or Thirtieth%
2028 \or Fortieth%
2029 \or Fiftieth%
2030 \or Sixtieth%
2031 \or Seventieth%
2032 \or Eightieth%
2033 \or Ninetieth%
2034 \fi
2035 }

```

The teens:

```

2036 \newcommand*\@@Teenthstringenglish}[1]{%
2037 \ifcase#1\relax
2038 Tenth%
2039 \or Eleventh%
2040 \or Twelfth%
2041 \or Thirteenth%
2042 \or Fourteenth%
2043 \or Fifteenth%
2044 \or Sixteenth%
2045 \or Seventeenth%
2046 \or Eighteenth%
2047 \or Nineteenth%
2048 \fi
2049 }

```

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.

```

2050 \newcommand*\@@Ordinalstringenglish}[2]{%
2051 \@strctr=#1\relax
2052 \ifnum#1>99999
2053 \PackageError{fmtcount}{Out of range}%
2054 {This macro only works for values less than 100000 (value given: \number\@strctr)}%
2055 \else
2056 \ifnum#1<0
2057 \PackageError{fmtcount}{Negative numbers not permitted}%
2058 {This macro does not work for negative numbers, however
2059 you can try typing "minus" first, and then pass the modulus of
2060 this number}%
2061 \fi
2062 \def#2{}%
2063 \fi
2064 \@strctr=#1\relax \divide\@strctr by 1000\relax
2065 \ifnum\@strctr>9\relax

```

```

#1 is greater or equal to 10000
2066 \divide\@strctr by 10
2067 \ifnum\@strctr>1\relax
2068 \let\@@fc@ordstr#2\relax
2069 \edef#2{\@@fc@ordstr\@tenstring{\@strctr}}%
2070 \@strctr=#1\relax
2071 \divide\@strctr by 1000\relax
2072 \@modulo{\@strctr}{10}%
2073 \ifnum\@strctr>0\relax
2074 \let\@@fc@ordstr#2\relax
2075 \edef#2{\@@fc@ordstr-\@unitstring{\@strctr}}%
2076 \fi
2077 \else
2078 \@strctr=#1\relax \divide\@strctr by 1000\relax
2079 \@modulo{\@strctr}{10}%
2080 \let\@@fc@ordstr#2\relax
2081 \edef#2{\@@fc@ordstr\@teenstring{\@strctr}}%
2082 \fi
2083 \@strctr=#1\relax \@modulo{\@strctr}{1000}%
2084 \ifnum\@strctr=0\relax
2085 \let\@@fc@ordstr#2\relax
2086 \edef#2{\@@fc@ordstr\ \@thousandth}%
2087 \else
2088 \let\@@fc@ordstr#2\relax
2089 \edef#2{\@@fc@ordstr\ \@thousand}%
2090 \fi
2091 \else
2092 \ifnum\@strctr>0\relax
2093 \let\@@fc@ordstr#2\relax
2094 \edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
2095 \@strctr=#1\relax \@modulo{\@strctr}{1000}%
2096 \let\@@fc@ordstr#2\relax
2097 \ifnum\@strctr=0\relax
2098 \edef#2{\@@fc@ordstr\ \@thousandth}%
2099 \else
2100 \edef#2{\@@fc@ordstr\ \@thousand}%
2101 \fi
2102 \fi
2103 \fi
2104 \@strctr=#1\relax \@modulo{\@strctr}{1000}%
2105 \divide\@strctr by 100
2106 \ifnum\@strctr>0\relax
2107 \ifnum#1>1000\relax
2108 \let\@@fc@ordstr#2\relax
2109 \edef#2{\@@fc@ordstr\ }%
2110 \fi
2111 \let\@@fc@ordstr#2\relax
2112 \edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
2113 \@strctr=#1\relax \@modulo{\@strctr}{100}%

```

```

2114 \let\@@fc@ordstr#2\relax
2115 \ifnum\@strctr=0\relax
2116   \edef#2{\@@fc@ordstr\ \@hundredth}%
2117 \else
2118   \edef#2{\@@fc@ordstr\ \@hundred}%
2119 \fi
2120 \fi
2121 \@strctr=#1\relax \@modulo{\@strctr}{100}%
2122 \ifnum#1>100\relax
2123   \ifnum\@strctr>0\relax
2124     \let\@@fc@ordstr#2\relax
2125     \edef#2{\@@fc@ordstr\ \@andname\ }%
2126   \fi
2127 \fi
2128 \ifnum\@strctr>19\relax
2129   \@tmpstrctr=\@strctr
2130   \divide\@strctr by 10\relax
2131   \@modulo{\@tmpstrctr}{10}%
2132   \let\@@fc@ordstr#2\relax
2133   \ifnum\@tmpstrctr=0\relax
2134     \edef#2{\@@fc@ordstr\@tenthstring{\@strctr}}%
2135   \else
2136     \edef#2{\@@fc@ordstr\@tenstring{\@strctr}}%
2137   \fi
2138   \@strctr=#1\relax \@modulo{\@strctr}{10}%
2139   \ifnum\@strctr>0\relax
2140     \let\@@fc@ordstr#2\relax
2141     \edef#2{\@@fc@ordstr-\@unitthstring{\@strctr}}%
2142   \fi
2143 \else
2144   \ifnum\@strctr<10\relax
2145     \ifnum\@strctr=0\relax
2146       \ifnum#1<100\relax
2147         \let\@@fc@ordstr#2\relax
2148         \edef#2{\@@fc@ordstr\@unitthstring{\@strctr}}%
2149       \fi
2150     \else
2151       \let\@@fc@ordstr#2\relax
2152       \edef#2{\@@fc@ordstr\@unitthstring{\@strctr}}%
2153     \fi
2154   \else
2155     \@modulo{\@strctr}{10}%
2156     \let\@@fc@ordstr#2\relax
2157     \edef#2{\@@fc@ordstr\@teenthstring{\@strctr}}%
2158   \fi
2159 \fi
2160 }

```


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

```
2161 \DeclareRobustCommand{\@OrdinalstringMenglish}[2]{%
2162 \let\@unitthstring=\@unitthstringenglish
2163 \let\@teenthstring=\@teenthstringenglish
2164 \let\@tenthstring=\@tenthstringenglish
2165 \let\@unitstring=\@unitstringenglish
2166 \let\@teenstring=\@teenstringenglish
2167 \let\@tenstring=\@tenstringenglish
2168 \def\@andname{and}%
2169 \def\@hundred{hundred}\def\@thousand{thousand}%
2170 \def\@hundredth{hundredth}\def\@thousandth{thousandth}%
2171 \@Ordinalstringenglish{#1}{#2}}
```

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

```
2172 \let\@OrdinalstringFenglish=\@OrdinalstringMenglish
2173 \let\@OrdinalstringNenglish=\@OrdinalstringMenglish
```

First letter of each word in upper case:

```
2174 \DeclareRobustCommand{\@OrdinalstringMenglish}[2]{%
2175 \let\@unitthstring=\@Unitthstringenglish
2176 \let\@teenthstring=\@Teenthstringenglish
2177 \let\@tenthstring=\@Tenthstringenglish
2178 \let\@unitstring=\@Unitstringenglish
2179 \let\@teenstring=\@Teenstringenglish
2180 \let\@tenstring=\@Tenstringenglish
2181 \def\@andname{and}%
2182 \def\@hundred{Hundred}\def\@thousand{Thousand}%
2183 \def\@hundredth{Hundredth}\def\@thousandth{Thousandth}%
2184 \@Ordinalstringenglish{#1}{#2}}
```

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

```
2185 \let\@OrdinalstringFenglish=\@OrdinalstringMenglish
2186 \let\@OrdinalstringNenglish=\@OrdinalstringMenglish
```

9.4.4 fc-francais.def

```
2187 \ProvidesFCLanguage{francais}[2012/06/18]
2188 \FCloadlang{french}
```

Set francais to be equivalent to french.

```
2189 \let\@ordinalMfrancais=\@ordinalMfrench
2190 \let\@ordinalFfrancais=\@ordinalFfrench
2191 \let\@ordinalNfrancais=\@ordinalNfrench
2192 \let\@numberstringMfrancais=\@numberstringMfrench
2193 \let\@numberstringFfrancais=\@numberstringFfrench
2194 \let\@numberstringNfrancais=\@numberstringNfrench
2195 \let\@NumberstringMfrancais=\@NumberstringMfrench
2196 \let\@NumberstringFfrancais=\@NumberstringFfrench
2197 \let\@NumberstringNfrancais=\@NumberstringNfrench
2198 \let\@ordinalstringMfrancais=\@ordinalstringMfrench
```

```

2199 \let\@ordinalstringFfrançais=\@ordinalstringFfrench
2200 \let\@ordinalstringNfrançais=\@ordinalstringNfrench
2201 \let\@ordinalstringMfrançais=\@ordinalstringMfrench
2202 \let\@ordinalstringFfrançais=\@ordinalstringFfrench
2203 \let\@ordinalstringNfrançais=\@ordinalstringNfrench

```

9.4.5 fc-french.def

Definitions for French.

```
2204 \ProvidesFCLanguage{french}[2012/10/24]
```

Package fcprefix is needed to format the prefix $\langle n \rangle$ in $\langle n \rangle$ illion or $\langle n \rangle$ illiard. Big numbers were developed based 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'.
2205 \def\fc@french@set@plural#1#2#3#4#5#6{%
2206   \ifthenelse{\equal{#2}{reformed}}{%
2207     \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{#3}%
2208   }{%
2209     \ifthenelse{\equal{#2}{traditional}}{%
2210       \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{#4}%
2211     }{%
2212       \ifthenelse{\equal{#2}{reformed o}}{%
2213         \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{#5}%
2214       }{%
2215         \ifthenelse{\equal{#2}{traditional o}}{%
2216           \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{#6}%
2217         }{%
2218           \ifthenelse{\equal{#2}{always}}{%
2219             \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{0}%
2220           }{%
2221             \ifthenelse{\equal{#2}{never}}{%
2222               \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{1}%
2223             }{%
2224               \ifthenelse{\equal{#2}{multiple}}{%
2225                 \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{2}%
2226               }{%
2227                 \ifthenelse{\equal{#2}{multiple g-last}}{%
2228                   \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{3}%
2229                 }{%
2230                   \ifthenelse{\equal{#2}{multiple l-last}}{%

```

```

2231         \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{4}%
2232     }{%
2233         \ifthenelse{\equal{#2}{multiple lng-last}}{%
2234             \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{5}%
2235         }{%
2236             \ifthenelse{\equal{#2}{multiple ng-last}}{%
2237                 \expandafter\def\csname fc@frenchoptions@#1@plural\endcsname{6}%
2238             }{%
2239                 \PackageError{fmtcount}{Unexpected argument}{%
2240                     ‘#2’ was unexpected: french option ‘#1 plural’ expects ‘reformed’
2241                     ‘reformed o’, ‘traditional o’, ‘always’, ‘never’, ‘multiple’, ‘mu
2242                     ‘multiple l-last’, ‘multiple lng-last’, or ‘multiple ng-last’.%
2243                 }
                }}}}}}}}}}}

```

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’.

```

2244 \def\@tempa#1#2#3{%
2245     \define@key{fcfrench}{#1 plural}[reformed]{%
2246         \fc@french@set@plural{#1}{##1}{#2}{#3}{#3}%
2247     }%
2248 }
2249 \@tempa{vingt}{4}{5}
2250 \@tempa{cent}{4}{5}
2251 \@tempa{mil}{0}{0}
2252 \@tempa{n-illion}{2}{6}
2253 \@tempa{n-illiard}{2}{6}

```

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

```

2254 \define@key{fcfrench}{all plural}[reformed]{%
2255     \csname KV@fcfrench@vingt plural\endcsname{#1}%
2256     \csname KV@fcfrench@cent plural\endcsname{#1}%
2257     \csname KV@fcfrench@mil plural\endcsname{#1}%
2258     \csname KV@fcfrench@n-illion plural\endcsname{#1}%
2259     \csname KV@fcfrench@n-illiard plural\endcsname{#1}%
2260 }

```

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

```

2261 \define@key{fcfrench}{dash or space}[reformed]{%
2262     \ifthenelse{\equal{#1}{traditional}}{%
2263         \let\fc@frenchoptions@supermillion@dos\space%
2264         \let\fc@frenchoptions@submillion@dos\space

```

```

2265 }{%
2266   \ifthenelse{\equal{#1}{reformed}}\or\equal{#1}{1990}}{%
2267     \let\fc@frenchoptions@supermillion@dos\space
2268     \def\fc@frenchoptions@submillion@dos{-}%
2269   }{%
2270     \ifthenelse{\equal{#1}{always}}{%
2271       \def\fc@frenchoptions@supermillion@dos{-}%
2272       \def\fc@frenchoptions@submillion@dos{-}%
2273     }{%
2274       \PackageError{fmtcount}{Unexpected argument}{%
2275         French option ‘dash or space’ expects ‘always’, ‘reformed’ or ‘traditional’
2276       }
2277     }%
2278   }%
2279 }%
2280 }

```

Option ‘scale’, can take 3 possible values:

- long for which $\langle n \rangle$ illions & $\langle n \rangle$ illiards are used with $10^{6 \times n} = 1 \langle n \rangle$ illion, and $10^{6 \times n + 3} = 1 \langle n \rangle$ illiard
- short for which $\langle n \rangle$ illions only are used with $10^{3 \times n + 3} = 1 \langle n \rangle$ illion
- recursive for which $10^{18} = \text{un milliard de milliards}$

```

2281 \define@key{fcfrench}{scale}[recursive]{%
2282   \ifthenelse{\equal{#1}{long}}{%
2283     \let\fc@poweroften\fc@pot@longscalefrench
2284   }{%
2285     \ifthenelse{\equal{#1}{recursive}}{%
2286       \let\fc@poweroften\fc@pot@recursivefrench
2287     }{%
2288       \ifthenelse{\equal{#1}{short}}{%
2289         \let\fc@poweroften\fc@pot@shortscalefrench
2290       }{%
2291         \PackageError{fmtcount}{Unexpected argument}{%
2292           French option ‘scale’ expects ‘long’, ‘recursive’ or ‘short’
2293         }
2294       }%
2295     }%
2296   }%
2297 }

```

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

- infinity in that case $\langle n \rangle$ illiard are never disabled,
- infty this is just a shorthand for ‘infinity’, and
- n any integer that is such that $n > 0$, and that $\forall k \in \mathbb{N}, k \geq n$, number $10^{6 \times k + 3}$ will be formatted as “mille $\langle n \rangle$ illions”

```

2298 \define@key{fcfrench}{n-illiard upto}[infinity]{%
2299   \ifthenelse{\equal{#1}{infinity}}{%
2300     \def\fc@longscale@nilliard@upto{0}%

```

```

2301 }{%
2302   \ifthenelse{\equal{#1}{infty}}{%
2303     \def\fc@longscale@milliard@upto{0}%
2304   }{%
2305     \if Q\ifnum9<1#1Q\fi\else
2306       \PackageError{fmtcount}{Unexpected argument}{%
2307         French option ‘milliard threshold’ expects ‘infinity’, or equivalently ‘infty’, or
2308         integer.}%
2309     \fi
2310     \def\fc@longscale@milliard@upto{#1}%
2311   }%
2312 }

```

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.

```

2313 \def\@tempa#1{%
2314   \define@key{fcfrench}{#1}[]{%
2315     \PackageError{fmtcount}{Unexpected argument}{French option with key ‘#1’ does not take
2316       any value}}%
2317   \expandafter\def\csname KV@fcfrench@#1@default\endcsname{%
2318     \def\fmtcount@french{#1}}%
2319 }%
2320 \@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.

```

2321 \define@key{fcfrench}{dialect}[france]{%
2322   \ifthenelse{\equal{#1}{france}}
2323     \or\equal{#1}{swiss}
2324     \or\equal{#1}{belgian}}{%
2325   \def\fmtcount@french{#1}}{%
2326   \PackageError{fmtcount}{Invalid value ‘#1’ to french option dialect key}
2327   {Option ‘french’ can only take the values ‘france’,
2328     ‘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’.

```

2329 \define@key{fcfrench}{mil plural mark}[le]{%
2330   \def\fc@frenchoptions@mil@plural@mark{#1}}

```

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

```

2331 \def\fc@UpperCaseFirstLetter#1#2\@nil{%
2332   \uppercase{#1}#2}
2333
2334 \def\fc@CaseIden#1\@nil{%
2335   #1%
2336 }

```

```

2337 \def\fc@UpperCaseAll#1\@nil{%
2338   \uppercase{#1}%
2339 }
2340
2341 \let\fc@case\fc@CaseIden
2342
\@ ordinalMfrench
2343 \newcommand*{\@ordinalMfrench}[2]{%
2344   \iffmtord@abbrv
2345   \edef#2{\number#1\relax\noexpand\fmtord{e}}%
2346   \else
2347     \ifnum#1=1\relax
2348       \edef#2{\number#1\relax\noexpand\fmtord{er}}%
2349     \else
2350       \edef#2{\number#1\relax\noexpand\fmtord{eme}}%
2351     \fi
2352   \fi}

\@ ordinalFfrench
2353 \newcommand*{\@ordinalFfrench}[2]{%
2354   \iffmtord@abbrv
2355   \edef#2{\number#1\relax\noexpand\fmtord{e}}%
2356   \else
2357     \ifnum#1=1 %
2358       \edef#2{\number#1\relax\noexpand\fmtord{i\`ere}}%
2359     \else
2360       \edef#2{\number#1\relax\noexpand\fmtord{i\`eme}}%
2361     \fi
2362   \fi}

  In French neutral gender and masculine gender are formally identical.
2363 \let\@ordinalNfrench\@ordinalMfrench

\@ @unitstringfrench
2364 \newcommand*{\@unitstringfrench}[1]{%
2365   \noexpand\fc@case
2366   \ifcase#1 %
2367   z\`ero%
2368   \or un%
2369   \or deux%
2370   \or trois%
2371   \or quatre%
2372   \or cinq%
2373   \or six%
2374   \or sept%
2375   \or huit%
2376   \or neuf%
2377   \fi
2378   \noexpand\@nil
2379 }

```

\@ @tenstringfrench

```
2380 \newcommand*{\@@tenstringfrench}[1]{%
2381 \noexpand\fc@case
2382 \ifcase#1 %
2383 \or dix%
2384 \or vingt%
2385 \or trente%
2386 \or quarante%
2387 \or cinquante%
2388 \or soixante%
2389 \or septante%
2390 \or huitante%
2391 \or nonante%
2392 \or cent%
2393 \fi
2394 \noexpand\@nil
2395 }
```

\@ @teenstringfrench

```
2396 \newcommand*{\@@teenstringfrench}[1]{%
2397 \noexpand\fc@case
2398 \ifcase#1 %
2399     dix%
2400 \or onze%
2401 \or douze%
2402 \or treize%
2403 \or quatorze%
2404 \or quinze%
2405 \or seize%
2406 \or dix\noexpand\@nil-\noexpand\fc@case sept%
2407 \or dix\noexpand\@nil-\noexpand\fc@case huit%
2408 \or dix\noexpand\@nil-\noexpand\fc@case neuf%
2409 \fi
2410 \noexpand\@nil
2411 }
```

\@ @seventiesfrench

```
2412 \newcommand*{\@@seventiesfrench}[1]{%
2413 \@tenstring{6}%
2414 \ifnum#1=1 %
2415 \fc@frenchoptions@submillion@dos\@andname\fc@frenchoptions@submillion@dos
2416 \else
2417 -%
2418 \fi
2419 \@teenstring{#1}%
2420 }
```

\@ @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".

```

2421 \newcommand*{\@eightiesfrench[1]}{
2422 \fc@case quatre\@nil-\noexpand\fc@case vingt%
2423 \ifnum#1>0 %
2424 \ifnum\fc@frenchoptions@vingt@plural=0 % vingt plural=always
2425 s%
2426 \fi
2427 \noexpand\@nil
2428 -\@unitstring{#1}%
2429 \else
2430 \ifcase\fc@frenchoptions@vingt@plural\space
2431 s% 0: always
2432 \or
2433 % 1: never
2434 \or
2435 s% 2: multiple
2436 \or
2437 % 3: multiple g-last
2438 \ifnum\count0=\count6\ifnum\count9=0 s\fi\fi
2439 \or
2440 % 4: multiple l-last
2441 \ifnum\count9=1 %
2442 \else
2443 s%
2444 \fi
2445 \or
2446 % 5: multiple lng-last
2447 \ifnum\count9=1 %
2448 \else
2449 \ifnum\count0>0 %
2450 s%
2451 \fi
2452 \fi
2453 \or
2454 % or 6: multiple ng-last
2455 \ifnum\count0>0 %
2456 s%
2457 \fi
2458 \fi
2459 \noexpand\@nil

```



```

2460 \fi
2461 }
2462 \newcommand*{\@@ninetiesfrench}[1]{%
2463 \fc@case quatre\@nil-\noexpand\fc@case vingt%
2464 \ifnum\fc@frenchoptions@vingt@plural=0 % vingt plural=always
2465 s%
2466 \fi
2467 \noexpand\@nil
2468 -\@teenstring{#1}%
2469 }
2470 \newcommand*{\@@seventiesfrenchswiss}[1]{%
2471 \@tenstring{7}%
2472 \ifnum#1=1\ \@andname\ \fi
2473 \ifnum#1>1-\fi
2474 \ifnum#1>0 \@unitstring{#1}\fi
2475 }
2476 \newcommand*{\@@eightiesfrenchswiss}[1]{%
2477 \@tenstring{8}%
2478 \ifnum#1=1\ \@andname\ \fi
2479 \ifnum#1>1-\fi
2480 \ifnum#1>0 \@unitstring{#1}\fi
2481 }
2482 \newcommand*{\@@ninetiesfrenchswiss}[1]{%
2483 \@tenstring{9}%
2484 \ifnum#1=1\ \@andname\ \fi
2485 \ifnum#1>1-\fi
2486 \ifnum#1>0 \@unitstring{#1}\fi
2487 }

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

2488 \newcommand*\fc@french@common{%
2489 \let\@unitstring=\@unitstringfrench
2490 \let\@teenstring=\@teenstringfrench
2491 \let\@tenstring=\@tenstringfrench
2492 \def\@hundred{cent}%
2493 \def\@andname{et}%
2494 }

2495 \DeclareRobustCommand{\@numberstringMfrenchswiss}[2]{%
2496 \let\fc@case\fc@CaseIden
2497 \fc@french@common
2498 \let\@seventies=\@seventiesfrenchswiss
2499 \let\@eighties=\@eightiesfrenchswiss
2500 \let\@nineties=\@ninetiesfrenchswiss
2501 \let\fc@nbrstr@preamble\@empty
2502 \let\fc@nbrstr@postamble\@empty
2503 \@numberstringfrench{#1}{#2}}
2504 \DeclareRobustCommand{\@numberstringMfrenchfrance}[2]{%
2505 \let\fc@case\fc@CaseIden

```

```

2506 \fc@french@common
2507 \let\@seventies=\@seventiesfrench
2508 \let\@eighties=\@eightiesfrench
2509 \let\@nineties=\@ninetiesfrench
2510 \let\fc@nbrstr@preamble\@empty
2511 \let\fc@nbrstr@postamble\@empty
2512 \@@numberstringfrench{#1}{#2}}
2513 \DeclareRobustCommand{\@numberstringMfrenchbelgian}[2]{%
2514 \let\fc@case\fc@CaseIden
2515 \fc@french@common
2516 \let\@seventies=\@seventiesfrenchswiss
2517 \let\@eighties=\@eightiesfrench
2518 \let\@nineties=\@ninetiesfrench
2519 \let\fc@nbrstr@preamble\@empty
2520 \let\fc@nbrstr@postamble\@empty
2521 \@@numberstringfrench{#1}{#2}}
2522 \let\@numberstringMfrench=\@numberstringMfrenchfrance
2523 \DeclareRobustCommand{\@numberstringFfrenchswiss}[2]{%
2524 \let\fc@case\fc@CaseIden
2525 \fc@french@common
2526 \let\@seventies=\@seventiesfrenchswiss
2527 \let\@eighties=\@eightiesfrenchswiss
2528 \let\@nineties=\@ninetiesfrenchswiss
2529 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
2530 \let\fc@nbrstr@postamble\@empty
2531 \@@numberstringfrench{#1}{#2}}
2532 \DeclareRobustCommand{\@numberstringFfrenchfrance}[2]{%
2533 \let\fc@case\fc@CaseIden
2534 \fc@french@common
2535 \let\@seventies=\@seventiesfrench
2536 \let\@eighties=\@eightiesfrench
2537 \let\@nineties=\@ninetiesfrench
2538 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
2539 \let\fc@nbrstr@postamble\@empty
2540 \@@numberstringfrench{#1}{#2}}
2541 \DeclareRobustCommand{\@numberstringFfrenchbelgian}[2]{%
2542 \let\fc@case\fc@CaseIden
2543 \fc@french@common
2544 \let\@seventies=\@seventiesfrenchswiss
2545 \let\@eighties=\@eightiesfrench
2546 \let\@nineties=\@ninetiesfrench
2547 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
2548 \let\fc@nbrstr@postamble\@empty
2549 \@@numberstringfrench{#1}{#2}}
2550 \let\@numberstringFfrench=\@numberstringFfrenchfrance
2551 \let\@ordinalstringNfrench\@ordinalstringMfrench
2552 \DeclareRobustCommand{\@NumberstringMfrenchswiss}[2]{%
2553 \let\fc@case\fc@UpperCaseFirstLetter
2554 \fc@french@common

```

```

2555 \let\@seventies=\@seventiesfrenchswiss
2556 \let\@eighties=\@eightiesfrenchswiss
2557 \let\@nineties=\@ninetiesfrenchswiss
2558 \let\fc@nbrstr@preamble\@empty
2559 \let\fc@nbrstr@postamble\@empty
2560 \@@numberstringfrench{#1}{#2}}
2561 \DeclareRobustCommand{\@NumberstringMfrenchfrance}[2]{%
2562 \let\fc@case\fc@UpperCaseFirstLetter
2563 \fc@french@common
2564 \let\@seventies=\@seventiesfrench
2565 \let\@eighties=\@eightiesfrench
2566 \let\@nineties=\@ninetiesfrench
2567 \let\fc@nbrstr@preamble\@empty
2568 \let\fc@nbrstr@postamble\@empty
2569 \@@numberstringfrench{#1}{#2}}
2570 \DeclareRobustCommand{\@NumberstringMfrenchbelgian}[2]{%
2571 \let\fc@case\fc@UpperCaseFirstLetter
2572 \fc@french@common
2573 \let\@seventies=\@seventiesfrenchswiss
2574 \let\@eighties=\@eightiesfrench
2575 \let\@nineties=\@ninetiesfrench
2576 \let\fc@nbrstr@preamble\@empty
2577 \let\fc@nbrstr@postamble\@empty
2578 \@@numberstringfrench{#1}{#2}}
2579 \let\@NumberstringMfrench=\@NumberstringMfrenchfrance
2580 \DeclareRobustCommand{\@NumberstringFfrenchswiss}[2]{%
2581 \let\fc@case\fc@UpperCaseFirstLetter
2582 \fc@french@common
2583 \let\@seventies=\@seventiesfrenchswiss
2584 \let\@eighties=\@eightiesfrenchswiss
2585 \let\@nineties=\@ninetiesfrenchswiss
2586 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
2587 \let\fc@nbrstr@postamble\@empty
2588 \@@numberstringfrench{#1}{#2}}
2589 \DeclareRobustCommand{\@NumberstringFfrenchfrance}[2]{%
2590 \let\fc@case\fc@UpperCaseFirstLetter
2591 \fc@french@common
2592 \let\@seventies=\@seventiesfrench
2593 \let\@eighties=\@eightiesfrench
2594 \let\@nineties=\@ninetiesfrench
2595 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
2596 \let\fc@nbrstr@postamble\@empty
2597 \@@numberstringfrench{#1}{#2}}
2598 \DeclareRobustCommand{\@NumberstringFfrenchbelgian}[2]{%
2599 \let\fc@case\fc@UpperCaseFirstLetter
2600 \fc@french@common
2601 \let\@seventies=\@seventiesfrenchswiss
2602 \let\@eighties=\@eightiesfrench
2603 \let\@nineties=\@ninetiesfrench

```

```

2604 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
2605 \let\fc@nbrstr@postamble\@empty
2606 @@numberstringfrench{#1}{#2}}
2607 \let\@NumberstringFfrench=\@NumberstringFfrenchfrance
2608 \let\@NumberstringNfrench\@NumberstringMfrench
2609 \DeclareRobustCommand{\@ordinalstringMfrenchswiss}[2]{%
2610 \let\fc@case\fc@CaseIden
2611 \let\fc@first=\fc@@firstfrench
2612 \fc@french@common
2613 \let\@seventies=\@seventiesfrenchswiss
2614 \let\@eighties=\@eightiesfrenchswiss
2615 \let\@nineties=\@ninetiesfrenchswiss
2616 @@ordinalstringfrench{#1}{#2}}%
2617 }
2618 \newcommand*\fc@@firstfrench{premier}
2619 \newcommand*\fc@@firstFfrench{premi\'ere}
2620 \DeclareRobustCommand{\@ordinalstringMfrenchfrance}[2]{%
2621 \let\fc@case\fc@CaseIden
2622 \let\fc@first=\fc@@firstfrench
2623 \fc@french@common
2624 \let\@seventies=\@seventiesfrench
2625 \let\@eighties=\@eightiesfrench
2626 \let\@nineties=\@ninetiesfrench
2627 @@ordinalstringfrench{#1}{#2}}
2628 \DeclareRobustCommand{\@ordinalstringMfrenchbelgian}[2]{%
2629 \let\fc@case\fc@CaseIden
2630 \let\fc@first=\fc@@firstfrench
2631 \fc@french@common
2632 \let\@seventies=\@seventiesfrench
2633 \let\@eighties=\@eightiesfrench
2634 \let\@nineties=\@ninetiesfrench
2635 @@ordinalstringfrench{#1}{#2}}%
2636 }
2637 \let\@ordinalstringMfrench=\@ordinalstringMfrenchfrance
2638 \DeclareRobustCommand{\@ordinalstringFfrenchswiss}[2]{%
2639 \let\fc@case\fc@CaseIden
2640 \let\fc@first=\fc@@firstFfrench
2641 \fc@french@common
2642 \let\@seventies=\@seventiesfrenchswiss
2643 \let\@eighties=\@eightiesfrenchswiss
2644 \let\@nineties=\@ninetiesfrenchswiss
2645 @@ordinalstringfrench{#1}{#2}}%
2646 }
2647 \DeclareRobustCommand{\@ordinalstringFfrenchfrance}[2]{%
2648 \let\fc@case\fc@CaseIden
2649 \let\fc@first=\fc@@firstFfrench
2650 \fc@french@common
2651 \let\@seventies=\@seventiesfrench
2652 \let\@eighties=\@eightiesfrench

```

```

2653 \let\@nineties=\@ninetiesfrench
2654 \@ordinalstringfrench{#1}{#2}%
2655 }
2656 \DeclareRobustCommand{\@ordinalstringFfrenchbelgian}[2]{%
2657 \let\fc@case\fc@CaseIden
2658 \let\fc@first=\fc@@firstFfrench
2659 \fc@french@common
2660 \let\@seventies=\@seventiesfrench
2661 \let\@eighties=\@eightiesfrench
2662 \let\@nineties=\@ninetiesfrench
2663 \@ordinalstringfrench{#1}{#2}%
2664 }
2665 \let\@ordinalstringFfrench=\@ordinalstringFfrenchfrance
2666 \let\@ordinalstringNfrench\@ordinalstringMfrench
2667 \DeclareRobustCommand{\@OrdinalstringMfrenchswiss}[2]{%
2668 \let\fc@case\fc@UpperCaseFirstLetter
2669 \let\fc@first=\fc@@firstfrench
2670 \fc@french@common
2671 \let\@seventies=\@seventiesfrenchswiss
2672 \let\@eighties=\@eightiesfrenchswiss
2673 \let\@nineties=\@ninetiesfrenchswiss
2674 \@ordinalstringfrench{#1}{#2}%
2675 }
2676 \DeclareRobustCommand{\@OrdinalstringMfrenchfrance}[2]{%
2677 \let\fc@case\fc@UpperCaseFirstLetter
2678 \let\fc@first=\fc@@firstfrench
2679 \fc@french@common
2680 \let\@seventies=\@seventiesfrench
2681 \let\@eighties=\@eightiesfrench
2682 \let\@nineties=\@ninetiesfrench
2683 \@ordinalstringfrench{#1}{#2}%
2684 }
2685 \DeclareRobustCommand{\@OrdinalstringMfrenchbelgian}[2]{%
2686 \let\fc@case\fc@UpperCaseFirstLetter
2687 \let\fc@first=\fc@@firstfrench
2688 \fc@french@common
2689 \let\@seventies=\@seventiesfrench
2690 \let\@eighties=\@eightiesfrench
2691 \let\@nineties=\@ninetiesfrench
2692 \@ordinalstringfrench{#1}{#2}%
2693 }
2694 \let\@OrdinalstringMfrench=\@OrdinalstringMfrenchfrance
2695 \DeclareRobustCommand{\@OrdinalstringFfrenchswiss}[2]{%
2696 \let\fc@case\fc@UpperCaseFirstLetter
2697 \let\fc@first=\fc@@firstfrench
2698 \fc@french@common
2699 \let\@seventies=\@seventiesfrenchswiss
2700 \let\@eighties=\@eightiesfrenchswiss
2701 \let\@nineties=\@ninetiesfrenchswiss

```

```

2702 \@@ordinalstringfrench{#1}{#2}%
2703 }
2704 \DeclareRobustCommand{\@OrdinalstringFfrenchfrance}[2]{%
2705 \let\fc@case\fc@UpperCaseFirstLetter
2706 \let\fc@first=\fc@@firstFfrench
2707 \fc@french@common
2708 \let\@seventies=\@@seventiesfrench
2709 \let\@eighties=\@@eightiesfrench
2710 \let\@nineties=\@@ninetiesfrench
2711 \@@ordinalstringfrench{#1}{#2}%
2712 }
2713 \DeclareRobustCommand{\@OrdinalstringFfrenchbelgian}[2]{%
2714 \let\fc@case\fc@UpperCaseFirstLetter
2715 \let\fc@first=\fc@@firstFfrench
2716 \fc@french@common
2717 \let\@seventies=\@@seventiesfrench
2718 \let\@eighties=\@@eightiesfrench
2719 \let\@nineties=\@@ninetiesfrench
2720 \@@ordinalstringfrench{#1}{#2}%
2721 }
2722 \let\@OrdinalstringFfrench=\@OrdinalstringFfrenchfrance
2723 \let\@OrdinalstringNfrench\@OrdinalstringMfrench

```

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

```

2724 \ifundefined{fc@@do@plural@mark}{\PackageError{fmtcount}{Duplicate definition}{Redefinit
2725 '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 $\langle n \rangle$ illiard, $\langle n \rangle$ illion, mil, cent or vingt, whichever is applicable, that is to say it is 1 when the considered objet is not multiplied, and 2 or more when it is multiplied,
\count6 input, counter giving the least weight of non zero digits in top level formatted number integral part, with rounding down to a multiple of 3,
\count10 input, counter giving the plural mark control option.

```

2726 \def\fc@@do@plural@mark#1{%
2727 \ifcase\count10 %
2728 #1% 0=always
2729 \or% 1=never
2730 \or% 2=multiple

```

```

2731     \ifnum\count1>1 %
2732         #1%
2733     \fi
2734 \or% 3= multiple g-last
2735     \ifnum\count1>1 %
2736         \ifnum\count0=\count6 %
2737             #1%
2738         \fi
2739     \fi
2740 \or% 4= multiple l-last
2741     \ifnum\count1>1 %
2742         \ifnum\count9=1 %
2743             \else
2744                 #1%
2745             \fi
2746         \fi
2747 \or% 5= multiple lng-last
2748     \ifnum\count1>1 %
2749         \ifnum\count9=1 %
2750             \else
2751                 \if\count0>\count6 %
2752                     #1%
2753                 \fi
2754             \fi
2755         \fi
2756 \or% 6= multiple ng-last
2757     \ifnum\count1>1 %
2758         \ifnum\count0>\count6 %
2759             #1%
2760         \fi
2761     \fi
2762 \fi
2763 }

```

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

```

2764 \@ifundefined{fc@@nbrstr@Fpreamble}{%
2765     \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
2766         'fc@@nbrstr@Fpreamble'}}

```

\fc @@nbrstr@Fpreamble

```

2767 \def\fc@@nbrstr@Fpreamble{%
2768     \fc@read@unit{\count1}{0}%
2769     \ifnum\count1=1 %
2770         \let\fc@case@save\fc@case
2771         \def\fc@case{\noexpand\fc@case}%
2772         \def\@nil{\noexpand\@nil}%
2773         \let\fc@nbrstr@postamble\fc@@nbrstr@Fpostamble
2774     \fi
2775 }

```

\fc @@nbrstr@Fpostamble

```

2776 \def\fc@@nbrstr@Fpostamble{%
2777   \let\fc@case\fc@case@save
2778   \expandafter\fc@get@last@word\expandafter{\@tempa}\@tempb\@tempc
2779   \def\@tempd{un}%
2780   \ifx\@tempc\@tempd
2781     \let\@tempc\@tempa
2782     \edef\@tempa{\@tempb\fc@case une\@nil}%
2783   \fi
2784 }
```

\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.

```

2785 \ifundefined{fc@@pot@longscalefrench}{%
2786   \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
2787     '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

```

2788 \def\fc@@pot@longscalefrench#1#2#3{%
2789   {%
```

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

```

2790   \edef\@tempb{\number#1}%
```

Let \count1 be the plural value.

```

2791   \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 .

```

2792   \count2\count0 %
2793   \divide\count2 by 6 %
2794   \count3\count2 %
2795   \multiply\count3 by 6 %
2796   \count3-\count3 %
2797   \advance\count3 by \count0 %
2798   \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)”.

```
2799 \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.

```
2800 \edef\@temph{%
```

```
2801 \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.

```
2802 1%
```

```
2803 \else
```

```
2804 \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`.

```
2805 \ifnum\fc@longscale@nilliard@upto=0 %
```

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

```
2806 2%
```

```
2807 \else
```

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

```
2808 \ifnum\count2>\fc@longscale@nilliard@upto
```

```
2809 1%
```

```
2810 \else
```

```
2811 2%
```

```
2812 \fi
```

```
2813 \fi
```

```
2814 \else
```

```
2815 2%
```

```
2816 \fi
```

```
2817 \fi
```

```
2818 }%
```

```
2819 \ifnum\@temph=1 %
```

Here 10^w is formatted as “mil(le)”.

```
2820 \count10=\fc@frenchoptions@mil@plural\space
```

```
2821 \edef\@tempe{%
```

```
2822 \noexpand\fc@case
```

```
2823 mil%
```

```
2824 \fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark
```

```
2825 \noexpand\@nil
```

```
2826 }%
```

```
2827 \else
```

```
2828 % weight >= 6
```

```
2829 \expandafter\fc@@latin@cardinal@prefix\expandafter{\the\count2}\@tempg
```

```

2830      % now form the xxx-illion(s) or xxx-illiard(s) word
2831      \ifnum\count3>2 %
2832          \toks10{illiard}%
2833          \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space
2834      \else
2835          \toks10{illion}%
2836          \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
2837      \fi
2838      \edef\@tempe{%
2839          \noexpand\fc@case
2840          \@tempg
2841          \the\toks10 %
2842          \fc@@do@plural@mark s%
2843          \noexpand\@nil
2844      }%
2845      \fi
2846      \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.

```

2847      \let\@tempe\@empty
2848      \def\@temph{0}%
2849      \fi
2850      \else

```

Case of $w = 0$.

```

2851      \let\@tempe\@empty
2852      \def\@temph{0}%
2853      \fi

```

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

```

2854      \expandafter\toks\expandafter1\expandafter{\@tempe}%
2855      \toks0{#2}%
2856      \edef\@tempa{\the\toks0 \@temph \def\noexpand#3{\the\toks1}}%
2857      \expandafter
2858      }\@tempa
2859 }

```

`\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.

```

2860 \ifundefined{fc@@pot@shortscalefrench}{\%
2861   \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
2862     '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

```
2863 \def\fc@@pot@shortscalefrench#1#2#3{%
```

```
2864   {%
```

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

```
2865   \edef\@tempb{\number#1}%
```

And let `\count1` be the plural value.

```
2866   \count1=\@tempb
```

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

```
2867   \count2\count0 %
```

```
2868   \divide\count2 by 3 %
```

```
2869   \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 $\langle nothing \rangle$, “mil(le)” or “ $\langle n \rangle$ illion(s)| $\langle n \rangle$ illiard(s)”.

```
2870   \ifnum\count0>0 % If weight>=3, i.e we do have to append thousand or n-illion(s)/n-illi
```

```
2871   \ifnum\count1>0 % we have at least one thousand/n-illion/n-illiard
```

```
2872   \ifnum\count2=0 %
```

```
2873   \def\@temph{1}%
```

```
2874   \count1=\fc@frenchoptions@mil@plural\space
```

```
2875   \edef\@tempe{%
```

```
2876   mil%
```

```
2877   \fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark
```

```
2878   }%
```

```
2879   \else
```

```
2880   \def\@temph{2}%
```

```
2881   % weight >= 6
```

```
2882   \expandafter\fc@@latin@cardinal@pexif\expandafter{\the\count2}\@tempg
```

```
2883   \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
```

```
2884   \edef\@tempe{%
```

```
2885   \noexpand\fc@case
```

```
2886   \@tempg
```

```
2887   illion%
```

```
2888   \fc@@do@plural@mark s%
```

```

2889         \noexpand\@nil
2890     }%
2891     \fi
2892 \else

```

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

```

2893     \def\@temph{0}%
2894     \let\@tempe\@empty
2895     \fi
2896 \else

```

Here $w = 0$.

```

2897     \def\@temph{0}%
2898     \let\@tempe\@empty
2899     \fi
2900 % now place into \cs{@tempa} the assignment of results \cs{@temph} and \cs{@tempe} to to \
2901 % \texttt{\#3} for further propagation after closing brace.
2902 %     \begin{macrocode}
2903     \expandafter\toks\expandafter1\expandafter{\@tempe}%
2904     \toks0{\#2}%
2905     \edef\@tempa{\the\toks0 \@temph \def\noexpand#3{\the\toks1}}%
2906     \expandafter
2907   }\@tempa
2908 }

```

`\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.

```

2909 \ifundefined{fc@@pot@recursivefrench}{\}%
2910 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
2911   ‘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

```

2912 \def\fc@@pot@recursivefrench#1#2#3{%
2913   {%

```

First the input arguments are saved into local objects: #1 and #2 are respectively saved into `\@tempa` and `\@tempb`.

```

2914     \edef\@tempb{\number#1}%
2915     \let\@tempa\@tempa

```

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

2916 \count1=\@tempb\space

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

```

2917 \ifnum\count1>0 %
2918 \count2\count0 %
2919 \divide\count2 by 9 %
2920 \advance\count2 by -1 %
2921 \let\@tempe\@empty
2922 \edef\@tempf{\fc@frenchoptions@supermillion@dos
2923 de\fc@frenchoptions@supermillion@dos\fc@case milliards\@nil}%
2924 \count11\count0 %
2925 \ifnum\count2>0 %
2926 \count3\count2 %
2927 \count3-\count3 %
2928 \multiply\count3 by 9 %
2929 \advance\count11 by \count3 %
2930 \loop
2931 % (\count2, \count3) <- (\count2 div 2, \count2 mod 2)
2932 \count3\count2 %
2933 \divide\count3 by 2 %
2934 \multiply\count3 by 2 %
2935 \count3-\count3 %
2936 \advance\count3 by \count2 %
2937 \divide\count2 by 2 %
2938 \ifnum\count3=1 %
2939 \let\@tempg\@tempe
2940 \edef\@tempe{\@tempg\@tempf}%
2941 \fi
2942 \let\@tempg\@tempf
2943 \edef\@tempf{\@tempg\@tempg}%
2944 \ifnum\count2>0 %
2945 \repeat
2946 \fi
2947 \divide\count11 by 3 %
2948 \ifcase\count11 % 0 .. 5
2949 % 0 => d milliard(s) (de milliards)*
2950 \def\@temph{2}%
2951 \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space
2952 \or % 1 => d mille milliard(s) (de milliards)*
2953 \def\@temph{1}%
2954 \count10=\fc@frenchoptions@mil@plural\space
2955 \or % 2 => d million(s) (de milliards)*
2956 \def\@temph{2}%
2957 \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
2958 \or % 3 => d milliard(s) (de milliards)*
2959 \def\@temph{2}%
2960 \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space

```

```

2961 \or % 4 => d mille milliards (de milliards)*
2962 \def\@temph{1}%
2963 \count10=\fc@frenchoptions@mil@plural\space
2964 \else % 5 => d million(s) (de milliards)*
2965 \def\@temph{2}%
2966 \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
2967 \fi
2968 \let\@tempg\@tempe
2969 \edef\@tempf{%
2970 \ifcase\count11 % 0 .. 5
2971 \or
2972 mil\fc@@do@plural@mark \fc@frenchoptions@mil@plural@mark
2973 \or
2974 million\fc@@do@plural@mark s%
2975 \or
2976 milliard\fc@@do@plural@mark s%
2977 \or
2978 mil\fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark
2979 \noexpand\@nil\fc@frenchoptions@supermillion@dos
2980 \noexpand\fc@case milliards% 4
2981 \or
2982 million\fc@@do@plural@mark s%
2983 \noexpand\@nil\fc@frenchoptions@supermillion@dos
2984 de\fc@frenchoptions@supermillion@dos\noexpand\fc@case milliards% 5
2985 \fi
2986 }%
2987 \edef\@tempe{%
2988 \ifx\@tempf\@empty\else
2989 \expandafter\fc@case\@tempf\@nil
2990 \fi
2991 \@tempg
2992 }%
2993 \else
2994 \def\@temph{0}%
2995 \let\@tempe\@empty
2996 \fi

```

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

```

2997 \expandafter\toks\expandafter1\expandafter{\@tempe}%
2998 \toks0{#2}%
2999 \edef\@tempa{\the\toks0 \@temph \def\noexpand#3{\the\toks1}}%
3000 \expandafter
3001 }\@tempa
3002 }

```

\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.

```
3003 \ifundefined{fc@muladdfrench}{-}{%
3004   \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
3005     'fc@muladdfrench'}}
```

Arguments as follows:

#2 input, plural indicator for number d

#3 input, formatted number d

#5 input, formatted number 10^w , i.e. power of ten which is multiplied by d

Implicit arguments from context:

\@tempa input, formatted number a

output, macro to which place the mul-add result

\count8 input, power type indicator for $10^{w'}$, where w' is a weight of a , this is an index in $[0..2]$ that reflects whether $10^{w'}$ is formatted by “mil(le)” — for index = 1 — or by “ $\langle n \rangle$ illion(s)| $\langle n \rangle$ illiard(s)” — for index = 2

\count9 input, power type indicator for 10^w , this is an index in $[0..2]$ that reflect whether the weight w of d is formatted by “metan-othing” — for index = 0, “mil(le)” — for index = 1 — or by “ $\langle n \rangle$ illion(s)| $\langle n \rangle$ illiard(s)” — for index = 2

```
3006 \def\fc@muladdfrench#1#2#3{%
3007   {%
```

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

```
3008   \edef\@tempc{#1}%
3009   \edef\@tempd{#2}%
3010   \edef\@tempf{#3}%
3011   \let\@tempc\@tempc
3012   \let\@tempd\@tempd
```

First we want to do the “multiplication” of $d \Rightarrow \text{\@tempd}$ and of $10^w \Rightarrow \text{\@tempf}$.

So, prior to this we do some preprocessing of $d \Rightarrow \text{\@tempd}$: 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”.

```
3013   \ifnum\@tempc=1 %
3014     \ifnum\count9=1 %
3015       \let\@tempd\@empty
3016     \fi
3017   \fi
```

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

```
3018   \edef\@tempg{%
3019     \@tempd
3020     \ifx\@tempd\@empty\else
3021       \ifx\@tempf\@empty\else
3022         \ifcase\count9 %
3023         \or
3024         \fc@frenchoptions@submillion@dos
```

```

3025         \or
3026         \fc@frenchoptions@supermillion@dos
3027     \fi
3028 \fi
3029 \fi
3030 \@tempf
3031 }%

```

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

```

3032 \edef\@temph{%
3033     \@tempa
3034     \ifx\@tempa\@empty\else
3035         \ifx\@tempg\@empty\else
3036             \ifcase\count8 %
3037                 \or
3038                 \fc@frenchoptions@submillion@dos
3039                 \or
3040                 \fc@frenchoptions@supermillion@dos
3041             \fi
3042         \fi
3043     \fi
3044     \@tempg
3045 }%

```

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

```

3046 \def\@tempb##1{\def\@tempa{\def\@tempa{##1}}}%
3047 \expandafter\@tempb\expandafter{\@temph}%
3048 \expandafter
3049 }\@tempa
3050 }%

```

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

```

3051 \ifundefined{fc@lthundredstringfrench}{}{%
3052     \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
3053         ‘fc@lthundredstringfrench’}}

```

The number to format is not passed as an argument to this macro, instead each digits of it is in a $\backslash\mathrm{fc}@digit@<w>$ macro after this number has been parsed. So the only thing that $\backslash\mathrm{fc}@lthundredstringfrench$ needs is to know $\langle w \rangle$ which is passed as $\backslash\mathrm{count}0$ for the less significant digit.

#1 input/output macro to which append the result

Implicit input arguments as follows:

$\backslash\mathrm{count}0$ 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.

```

3054 \def\fc@lthundredstringfrench#1{%
3055     {%

```


First save arguments into local temporary macro.

```
3056 \let\@tempc#1%
```

Read units d_w to \count1.

```
3057 \fc@read@unit{\count1}{\count0}%
```

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

```
3058 \count3\count0 %
```

```
3059 \advance\count3 1 %
```

```
3060 \fc@read@unit{\count2}{\count3}%
```

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

```
3061 \edef\@tempa{%
```

```
3062 \@tempc
```

```
3063 \ifnum\count2>1 %
```

```
3064 % 20 .. 99
```

```
3065 \ifnum\count2>6 %
```

```
3066 % 70 .. 99
```

```
3067 \ifnum\count2<8 %
```

```
3068 % 70 .. 79
```

```
3069 \@seventies{\count1}%
```

```
3070 \else
```

```
3071 % 80..99
```

```
3072 \ifnum\count2<9 %
```

```
3073 % 80 .. 89
```

```
3074 \@eighties{\count1}%
```

```
3075 \else
```

```
3076 % 90 .. 99
```

```
3077 \@nineties{\count1}%
```

```
3078 \fi
```

```
3079 \fi
```

```
3080 \else
```

```
3081 % 20..69
```

```
3082 \@tenstring{\count2}%
```

```
3083 \ifnum\count1>0 %
```

```
3084 % x1 .. x0
```

```
3085 \ifnum\count1=1 %
```

```
3086 % x1
```

```
3087 \fc@frenchoptions@submillion@dos\@andname\fc@frenchoptions@submillion@dos
```

```
3088 \else
```

```
3089 % x2 .. x9
```

```
3090 -%
```

```
3091 \fi
```

```
3092 \@unitstring{\count1}%
```

```
3093 \fi
```

```
3094 \fi
```

```
3095 \else
```

```
3096 % 0 .. 19
```

```
3097 \ifnum\count2=0 % when tens = 0
```

```
3098 % 0 .. 9
```

```

3099         \ifnum\count1=0 % when units = 0
3100         % \count3=1 when #1 = 0, i.e. only for the unit of the top level number
3101         \ifnum\count3=1 %
3102         \ifnum\fc@max@weight=0 %
3103         \@unitstring{0}%
3104         \fi
3105         \fi
3106         \else
3107         % 1 .. 9
3108         \@unitstring{\count1}%
3109         \fi
3110         \else
3111         % 10 .. 19
3112         \@teenstring{\count1}%
3113         \fi
3114         \fi
3115     }%

```

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

```

3116     \def\@tempb##1{\def\@tempa{\def#1{##1}}}%
3117     \expandafter\@tempb\expandafter{\@tempa}%
3118     \expandafter
3119 } \@tempa
3120 }

```

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

```

3121 \ifundefined{fc@ltthousandstringfrench}{\{}%
3122 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
3123 '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)

```

3124 \def\fc@ltthousandstringfrench#1{%
3125     {%

```

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

```

3126     \count4\count0 %
3127     \advance\count4 by 2 %
3128     \fc@read@unit{\count2 }{\count4 }%

```

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

```

3129     \advance\count4 by -1 %
3130     \count3\count4 %

```

```

3131 \advance\count3 by -1 %
3132 \fc@check@nonzeros{\count3 }{\count4 }\@tempa

  Compute plural mark of 'cent' into \@temps.

3133 \edef\@temps{%
3134   \ifcase\fc@frenchoptions@cent@plural\space
3135   % 0 => always
3136   s%
3137   \or
3138   % 1 => never
3139   \or
3140   % 2 => multiple
3141   \ifnum\count2>1s\fi
3142   \or
3143   % 3 => multiple g-last
3144   \ifnum\count2>1 \ifnum\@tempa=0 \ifnum\count0=\count6s\fi\fi\fi
3145   \or
3146   % 4 => multiple l-last
3147   \ifnum\count2>1 \ifnum\@tempa=0 \ifnum\count9=0s\else\ifnum\count9=2s\fi\fi\fi\fi
3148   \fi
3149 }%
3150 % compute spacing after cent(s?) into \@tempb
3151 \expandafter\let\expandafter\@tempb
3152 \ifnum\@tempa>0 \fc@frenchoptions@submillion@dos\else\@empty\fi
3153 % now place into \@tempa the hundreds
3154 \edef\@tempa{%
3155   \ifnum\count2=0 %
3156   \else
3157   \ifnum\count2=1 %
3158   \expandafter\fc@case\@hundred\@nil
3159   \else
3160   \@unitstring{\count2}\fc@frenchoptions@submillion@dos
3161   \noexpand\fc@case\@hundred\@temps\noexpand\@nil
3162   \fi
3163   \@tempb
3164   \fi
3165 }%
3166 % now append to \@tempa the ten and unit
3167 \fc@lthundredstringfrench\@tempa

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

3168 \def\@tempb##1{\def\@tempa{\def#1{##1}}}%
3169 \expandafter\@tempb\expandafter{\@tempa}%
3170 \expandafter
3171 }\@tempa
3172 }

```

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

```

3173 \ifundefined{@@numberstringfrench}{}%
3174 \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
3175 \def\@@numberstringfrench#1#2{%
3176 {%

```

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

```

3177 \edef\@tempa{#1}%
3178 \expandafter\fc@number@parser\expandafter{\@tempa}%
3179 \ifnum\fc@min@weight<0 %
3180 \PackageError{fmtcount}{Out of range}%
3181 {This macro does not work with fractional numbers}%
3182 \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’.

```

3183 \edef\@tempa{\ifcase\fc@sign@case\space\or\fc@case plus\@nil\or\fc@case moins\@nil\fi}%
3184 \fc@nbrstr@preamble
3185 \fc@@nbrstrfrench@inner
3186 \fc@nbrstr@postamble

```

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

```

3187 \def\@tempb##1{\def\@tempa{\def#2{##1}}}%
3188 \expandafter\@tempb\expandafter{\@tempa}%
3189 \expandafter
3190 }\@tempa
3191 }

```

\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.

```

3192 \def\fc@@nbrstrfrench@inner{%

```

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

```

3193 \count0=\fc@max@weight
3194 \divide\count0 by 3 %
3195 \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.

```

3196 \fc@intpart@find@last{\count5 }%
3197 \count6\count5 %
3198 \divide\count6 3 %

```

```

3199 \multiply\count6 3 %
3200 \count8=0 %
3201 \loop

```

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

```

3202 \count1\count0 %
3203 \advance\count1 by 2 %
3204 \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.

```

3205 \fc@powerof ten \@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

```

3206 \fc@lthousandstringfrench \@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 .

```

3207 \fc@muladdfrench \@tempt \@tempd \@tempb

```

Then iterate.

```

3208 \count8\count9 %
3209 \advance\count0 by -3 %
3210 \ifnum\count6>\count0 \else
3211 \repeat
3212 }

```

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

```

3213 \ifundefined{@@ordinalstringfrench}{\}%
3214 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
3215 '@@ordinalstringfrench'}}

```

Arguments are as follows:

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

```

3216 \def\@@ordinalstringfrench#1#2{%
3217 {%

```

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

```

3218 \edef\@tempa{#1}%
3219 \expandafter\fc@number@parser\expandafter{\@tempa}%
3220 \ifnum\fc@min@weight<0 %
3221 \PackageError{fmtcount}{Out of range}%
3222 {This macro does not work with fractional numbers}%
3223 \fi

```

```

3224 \ifnum\fc@sign@case>0 %
3225 \PackageError{fmtcount}{Out of range}%
3226 {This macro does with negative or explicitly marked as positive numbers}%
3227 \fi

```

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

```

3228 \ifnum\fc@max@weight=0 %
3229 \ifnum\csname fc@digit@0\endcsname=1 %
3230 \count0=1 %
3231 \else
3232 \count0=0 %
3233 \fi
3234 \else
3235 \count0=0 %
3236 \fi
3237 \ifnum\count0=1 %
3238 \edef\@tempa{\expandafter\fc@case\fc@first\@nil}%
3239 \else

```

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

```

3240 \def\@tempa##1{%
3241 \expandafter\edef\csname fc@frenchoptions@##1@plural\endcsname{%
3242 \ifcase\csname fc@frenchoptions@##1@plural\endcsname\space
3243 0% 0: always => always
3244 \or
3245 1% 1: never => never
3246 \or
3247 6% 2: multiple => multiple ng-last
3248 \or
3249 1% 3: multiple g-last => never
3250 \or
3251 5% 4: multiple l-last => multiple lng-last
3252 \or
3253 5% 5: multiple lng-last => multiple lng-last
3254 \or
3255 6% 6: multiple ng-last => multiple ng-last
3256 \fi
3257 }%
3258 }%
3259 \@tempa{vingt}%
3260 \@tempa{cent}%
3261 \@tempa{mil}%
3262 \@tempa{n-illion}%
3263 \@tempa{n-illiard}%

```

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

```

3264 \let\fc@case@save\fc@case
3265 \def\fc@case{\noexpand\fc@case}%
3266 \def\@nil{\noexpand\@nil}%

```

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

```
3267 \let\@tempa\@empty
3268 \fc@nbrstrfrench@inner
```

Now restore \fc@case

```
3269 \let\fc@case\fc@case@save
```

Now we add the “ième” ending

```
3270 \expandafter\fc@get@last@word\expandafter{\@tempa}\@tempb\@tempc
3271 \expandafter\fc@get@last@letter\expandafter{\@tempc}\@tempd\@tempe
3272 \def\@tempf{e}%
3273 \ifx\@tempe\@tempf
3274 \edef\@tempa{\@tempb\expandafter\fc@case\@tempd i\'eme\@nil}%
3275 \else
3276 \def\@tempf{q}%
3277 \ifx\@tempe\@tempf
3278 \edef\@tempa{\@tempb\expandafter\fc@case\@tempd qui\'eme\@nil}%
3279 \else
3280 \def\@tempf{f}%
3281 \ifx\@tempe\@tempf
3282 \edef\@tempa{\@tempb\expandafter\fc@case\@tempd vi\'eme\@nil}%
3283 \else
3284 \edef\@tempa{\@tempb\expandafter\fc@case\@tempc i\'eme\@nil}%
3285 \fi
3286 \fi
3287 \fi
3288 \fi
```

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

```
3289 \def\@tempb##1{\def\@tempa{\def#2{##1}}}%
3290 \expandafter\@tempb\expandafter{\@tempa}%
3291 \expandafter
3292 }\@tempa
3293 }
```

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

```
3294 \newcommand*\fc@frenchoptions@setdefaults{%
3295 \csname KV@fcfrench@all plural\endcsname{reformed}%
3296 \def\fc@frenchoptions@submillion@dos{-}%
3297 \let\fc@frenchoptions@supermillion@dos\space
3298 \let\fc@u@in@duo\@empty% Could be ‘u’
3299 % \let\fc@poweroften\fc@@pot@longscalefrench
3300 \let\fc@poweroften\fc@@pot@recursivefrench
3301 \def\fc@longscale@nilliard@upto{0}% infinity
3302 \def\fc@frenchoptions@mil@plural@mark{le}%
3303 }
3304 \fc@frenchoptions@setdefaults
```

9.4.6 fc-frenchb.def

```
3305 \ProvidesFCLanguage{frenchb}[2012/06/18]
3306 \FCloadlang{french}
```

Set frenchb to be equivalent to french.

```
3307 \let\@ordinalMfrenchb=\@ordinalMfrench
3308 \let\@ordinalFfrenchb=\@ordinalFfrench
3309 \let\@ordinalNfrenchb=\@ordinalNfrench
3310 \let\@numberstringMfrenchb=\@numberstringMfrench
3311 \let\@numberstringFfrenchb=\@numberstringFfrench
3312 \let\@numberstringNfrenchb=\@numberstringNfrench
3313 \let\@NumberstringMfrenchb=\@NumberstringMfrench
3314 \let\@NumberstringFfrenchb=\@NumberstringFfrench
3315 \let\@NumberstringNfrenchb=\@NumberstringNfrench
3316 \let\@ordinalstringMfrenchb=\@ordinalstringMfrench
3317 \let\@ordinalstringFfrenchb=\@ordinalstringFfrench
3318 \let\@ordinalstringNfrenchb=\@ordinalstringNfrench
3319 \let\@OrdinalstringMfrenchb=\@OrdinalstringMfrench
3320 \let\@OrdinalstringFfrenchb=\@OrdinalstringFfrench
3321 \let\@OrdinalstringNfrenchb=\@OrdinalstringNfrench
```

9.4.7 fc-german.def

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

```
3322 \ProvidesFCLanguage{german}[2012/06/18]
```

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

```
3323 \newcommand{\@ordinalMgerman}[2]{%
3324 \edef#2{\number#1\relax.}}
```

Feminine:

```
3325 \newcommand{\@ordinalFgerman}[2]{%
3326 \edef#2{\number#1\relax.}}
```

Neuter:

```
3327 \newcommand{\@ordinalNgerman}[2]{%
3328 \edef#2{\number#1\relax.}}
```

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

```
3329 \newcommand{\@@unitstringgerman}[1]{%
3330 \ifcase#1%
3331 null%
3332 \or eins%
3333 \or zwei%
3334 \or drei%
3335 \or vier%
3336 \or f\"unf%
```



```

3337 \or sechs%
3338 \or sieben%
3339 \or acht%
3340 \or neun%
3341 \fi
3342 }

```

Tens (argument must go from 1 to 10):

```

3343 \newcommand{\@@tenstringgerman}[1]{%
3344 \ifcase#1%
3345 \or zehn%
3346 \or zwanzig%
3347 \or drei{\ss}ig%
3348 \or vierzig%
3349 \or f\unfzig%
3350 \or sechzig%
3351 \or siebzig%
3352 \or achtzig%
3353 \or neunzig%
3354 \or einhundert%
3355 \fi
3356 }

```

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

```

3357 \providecommand*\einhundert{\einhundert}
3358 \providecommand*\eintausend{\eintausend}

```

Teens:

```

3359 \newcommand{\@@teenstringgerman}[1]{%
3360 \ifcase#1%
3361 zehn%
3362 \or elf%
3363 \or zw\olf%
3364 \or dreizehn%
3365 \or vierzehn%
3366 \or f\unfzehn%
3367 \or sechzehn%
3368 \or siebzehn%
3369 \or achtzehn%
3370 \or neunzehn%
3371 \fi
3372 }

```

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

```

3373 \DeclareRobustCommand{\@numberstringMgerman}[2]{%
3374 \let\@unitstring=\@unitstringgerman
3375 \let\@teenstring=\@teenstringgerman
3376 \let\@tenstring=\@tenstringgerman
3377 \@@numberstringgerman{#1}{#2}}

```

Feminine and neuter forms:

```
3378 \let\@numberstringFgerman=\@numberstringMgerman
3379 \let\@numberstringNgerman=\@numberstringMgerman
```

As above, but initial letters in upper case:

```
3380 \DeclareRobustCommand{\@NumberstringMgerman}[2]{%
3381   \@numberstringMgerman{#1}{\@num@str}%
3382   \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
3383 }
```

Feminine and neuter form:

```
3384 \let\@NumberstringFgerman=\@NumberstringMgerman
3385 \let\@NumberstringNgerman=\@NumberstringMgerman
```

As above, but for ordinals.

```
3386 \DeclareRobustCommand{\@OrdinalstringMgerman}[2]{%
3387 \let\@unitthstring=\@unitthstringMgerman
3388 \let\@teenthstring=\@teenthstringMgerman
3389 \let\@tenthstring=\@tenthstringMgerman
3390 \let\@unitstring=\@unitstringgerman
3391 \let\@teenstring=\@teenstringgerman
3392 \let\@tenstring=\@tenstringgerman
3393 \def\@thousandth{tausendster}%
3394 \def\@hundredth{hundertster}%
3395 \@@ordinalstringgerman{#1}{#2}}
```

Feminine form:

```
3396 \DeclareRobustCommand{\@OrdinalstringFgerman}[2]{%
3397 \let\@unitthstring=\@unitthstringFgerman
3398 \let\@teenthstring=\@teenthstringFgerman
3399 \let\@tenthstring=\@tenthstringFgerman
3400 \let\@unitstring=\@unitstringgerman
3401 \let\@teenstring=\@teenstringgerman
3402 \let\@tenstring=\@tenstringgerman
3403 \def\@thousandth{tausendste}%
3404 \def\@hundredth{hundertste}%
3405 \@@ordinalstringgerman{#1}{#2}}
```

Neuter form:

```
3406 \DeclareRobustCommand{\@OrdinalstringNgerman}[2]{%
3407 \let\@unitthstring=\@unitthstringNgerman
3408 \let\@teenthstring=\@teenthstringNgerman
3409 \let\@tenthstring=\@tenthstringNgerman
3410 \let\@unitstring=\@unitstringgerman
3411 \let\@teenstring=\@teenstringgerman
3412 \let\@tenstring=\@tenstringgerman
3413 \def\@thousandth{tausendstes}%
3414 \def\@hundredth{hunderstes}%
3415 \@@ordinalstringgerman{#1}{#2}}
```

As above, but with initial letters in upper case.

```

3416 \DeclareRobustCommand{\@OrdinalstringMgerman}[2]{%
3417 \@ordinalstringMgerman{#1}{\@num@str}%
3418 \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
3419 }

```

Feminine form:

```

3420 \DeclareRobustCommand{\@OrdinalstringFgerman}[2]{%
3421 \@ordinalstringFgerman{#1}{\@num@str}%
3422 \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
3423 }

```

Neuter form:

```

3424 \DeclareRobustCommand{\@OrdinalstringNgerman}[2]{%
3425 \@ordinalstringNgerman{#1}{\@num@str}%
3426 \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
3427 }

```

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

```

3428 \newcommand{\@unitthstringMgerman}[1]{%
3429 \ifcase#1%
3430 nullter%
3431 \or erster%
3432 \or zweiter%
3433 \or dritter%
3434 \or vierter%
3435 \or f\unfter%
3436 \or sechster%
3437 \or siebter%
3438 \or achter%
3439 \or neunter%
3440 \fi
3441 }

```

Tens:

```

3442 \newcommand{\@tenthstringMgerman}[1]{%
3443 \ifcase#1%
3444 \or zehnter%
3445 \or zwanzigster%
3446 \or drei{\ss}igster%
3447 \or vierzigster%
3448 \or f\unfzigster%
3449 \or sechzigster%
3450 \or siebzigster%
3451 \or achtzigster%
3452 \or neunzigster%
3453 \fi
3454 }

```

Teens:

```

3455 \newcommand{\@teenthstringMgerman}[1]{%

```

```

3456 \ifcase#1%
3457 zehnter%
3458 \or elfter%
3459 \or zw\"olfter%
3460 \or dreizehnter%
3461 \or vierzehnter%
3462 \or f\"unfzehnter%
3463 \or sechzehnter%
3464 \or siebzehnter%
3465 \or achtzehnter%
3466 \or neunzehnter%
3467 \fi
3468 }

```

Units (feminine):

```

3469 \newcommand{\@@unitthstringFgerman}[1]{%
3470 \ifcase#1%
3471 nullte%
3472 \or erste%
3473 \or zweite%
3474 \or dritte%
3475 \or vierte%
3476 \or f\"unfte%
3477 \or sechste%
3478 \or siebte%
3479 \or achte%
3480 \or neunte%
3481 \fi
3482 }

```

Tens (feminine):

```

3483 \newcommand{\@@tenthstringFgerman}[1]{%
3484 \ifcase#1%
3485 \or zehnte%
3486 \or zwanzigste%
3487 \or drei{\ss}igste%
3488 \or vierzigste%
3489 \or f\"unzigste%
3490 \or sechzigste%
3491 \or siebzigste%
3492 \or achtzigste%
3493 \or neunzigste%
3494 \fi
3495 }

```

Teens (feminine)

```

3496 \newcommand{\@@teenthstringFgerman}[1]{%
3497 \ifcase#1%
3498 zehnte%
3499 \or elfte%
3500 \or zw\"olfte%

```

```

3501 \or dreizehnte%
3502 \or vierzehnte%
3503 \or f\"unfzehnte%
3504 \or sechzehnte%
3505 \or siebzehnte%
3506 \or achtzehnte%
3507 \or neunzehnte%
3508 \fi
3509 }

```

Units (neuter):

```

3510 \newcommand{\@@unitthstringNgerman}[1]{%
3511 \ifcase#1%
3512 nulltes%
3513 \or erstes%
3514 \or zweites%
3515 \or drittes%
3516 \or viertes%
3517 \or f\"unftes%
3518 \or sechstes%
3519 \or siebtes%
3520 \or achttes%
3521 \or neuntes%
3522 \fi
3523 }

```

Tens (neuter):

```

3524 \newcommand{\@@tenthstringNgerman}[1]{%
3525 \ifcase#1%
3526 \or zehntes%
3527 \or zwanzigstes%
3528 \or drei{\ss}igstes%
3529 \or vierzigstes%
3530 \or f\"unfzigstes%
3531 \or sechzigstes%
3532 \or siebzigstes%
3533 \or achtzigstes%
3534 \or neunzigstes%
3535 \fi
3536 }

```

Teens (neuter)

```

3537 \newcommand{\@@teenthstringNgerman}[1]{%
3538 \ifcase#1%
3539 zehntes%
3540 \or elftes%
3541 \or zw\"olftes%
3542 \or dreizehntes%
3543 \or vierzehntes%
3544 \or f\"unfzehntes%
3545 \or sechzehntes%

```

```

3546 \or siebzehntes%
3547 \or achtzehntes%
3548 \or neunzehntes%
3549 \fi
3550 }

```

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

```

3551 \newcommand{\@@numberunderhundredgerman}[2]{%
3552 \ifnum#1<10\relax
3553   \ifnum#1>0\relax
3554     \eappto#2{\@unitstring{#1}}%
3555   \fi
3556 \else
3557   \@tmpstrctr=#1\relax
3558   \@modulo{\@tmpstrctr}{10}%
3559   \ifnum#1<20\relax
3560     \eappto#2{\@teenstring{\@tmpstrctr}}%
3561   \else
3562     \ifnum\@tmpstrctr=0\relax
3563     \else
3564       \eappto#2{\@unitstring{\@tmpstrctr}und}%
3565     \fi
3566     \@tmpstrctr=#1\relax
3567     \divide\@tmpstrctr by 10\relax
3568     \eappto#2{\@tenstring{\@tmpstrctr}}%
3569   \fi
3570 \fi
3571 }

```

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

```

3572 \newcommand{\@@numberstringgerman}[2]{%
3573 \ifnum#1>99999\relax
3574   \PackageError{fmtcount}{Out of range}%
3575   {This macro only works for values less than 100000}%
3576 \else
3577   \ifnum#1<0\relax
3578     \PackageError{fmtcount}{Negative numbers not permitted}%
3579     {This macro does not work for negative numbers, however
3580     you can try typing "minus" first, and then pass the modulus of
3581     this number}%
3582   \fi
3583 \fi
3584 \def#2{}%
3585 \@strctr=#1\relax \divide\@strctr by 1000\relax
3586 \ifnum\@strctr>1\relax

```

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

```

3587 \@@numberunderhundredgerman{\@strctr}{#2}%

```

```

3588 \appto#2{tausend}%
3589 \else
    #1 lies in range [1000,1999]
3590 \ifnum \@strctr=1\relax
3591 \eappto#2{\eintausend}%
3592 \fi
3593 \fi
3594 \@strctr=#1\relax
3595 \@modulo{\@strctr}{1000}%
3596 \divide \@strctr by 100\relax
3597 \ifnum \@strctr>1\relax
    now dealing with number in range [200,999]
3598 \eappto#2{\@unitstring{\@strctr}hundert}%
3599 \else
3600 \ifnum \@strctr=1\relax
    dealing with number in range [100,199]
3601 \ifnum #1>1000\relax
    if original number > 1000, use einhundert
3602 \appto#2{einhundert}%
3603 \else
    otherwise use \einhundert
3604 \eappto#2{\einhundert}%
3605 \fi
3606 \fi
3607 \fi
3608 \@strctr=#1\relax
3609 \@modulo{\@strctr}{100}%
3610 \ifnum #1=0\relax
3611 \def#2{null}%
3612 \else
3613 \ifnum \@strctr=1\relax
3614 \appto#2{eins}%
3615 \else
3616 \@@numberunderhundredgerman{\@strctr}{#2}%
3617 \fi
3618 \fi
3619 }

    As above, but for ordinals
3620 \newcommand{\@@numberunderhundredthgerman}[2]{%
3621 \ifnum #1<10\relax
3622 \eappto#2{\@unitthstring{#1}}%
3623 \else
3624 \@tmpstrctr=#1\relax
3625 \@modulo{\@tmpstrctr}{10}%
3626 \ifnum #1<20\relax
3627 \eappto#2{\@teenthsstring{\@tmpstrctr}}%

```

```

3628 \else
3629 \ifnum \@tmpstrctr=0\relax
3630 \else
3631 \eappto#2{\@unitstring{\@tmpstrctr}und}%
3632 \fi
3633 \@tmpstrctr=#1\relax
3634 \divide \@tmpstrctr by 10\relax
3635 \eappto#2{\@tenthstring{\@tmpstrctr}}%
3636 \fi
3637 \fi
3638 }

3639 \newcommand{\@@ordinalstringgerman}[2]{%
3640 \ifnum#1>99999\relax
3641 \PackageError{fmtcount}{Out of range}%
3642 {This macro only works for values less than 100000}%
3643 \else
3644 \ifnum#1<0\relax
3645 \PackageError{fmtcount}{Negative numbers not permitted}%
3646 {This macro does not work for negative numbers, however
3647 you can try typing "minus" first, and then pass the modulus of
3648 this number}%
3649 \fi
3650 \fi
3651 \def#2{}%
3652 \@strctr=#1\relax \divide \@strctr by 1000\relax
3653 \ifnum \@strctr>1\relax

#1 is  $\geq 2000$ , \@strctr now contains the number of thousands
3654 \@@numberunderhundredgerman{\@strctr}{#2}%

is that it, or is there more?
3655 \@tmpstrctr=#1\relax \@modulo{\@tmpstrctr}{1000}%
3656 \ifnum \@tmpstrctr=0\relax
3657 \eappto#2{\@thousandth}%
3658 \else
3659 \appto#2{tausend}%
3660 \fi
3661 \else

#1 lies in range [1000,1999]
3662 \ifnum \@strctr=1\relax
3663 \ifnum#1=1000\relax
3664 \eappto#2{\@thousandth}%
3665 \else
3666 \eappto#2{eintausend}%
3667 \fi
3668 \fi
3669 \fi
3670 \@strctr=#1\relax
3671 \@modulo{\@strctr}{1000}%

```



```

3672 \divide \@strctr by 100\relax
3673 \ifnum \@strctr>1\relax
    now dealing with number in range [200,999] is that it, or is there more?
3674 \@tmpstrctr=#1\relax \@modulo{\@tmpstrctr}{100}%
3675 \ifnum \@tmpstrctr=0\relax
3676 \ifnum \@strctr=1\relax
3677 \eappto#2{\@hundredth}%
3678 \else
3679 \eappto#2{\@unitstring{\@strctr}\@hundredth}%
3680 \fi
3681 \else
3682 \eappto#2{\@unitstring{\@strctr}hundert}%
3683 \fi
3684 \else
3685 \ifnum \@strctr=1\relax
    dealing with number in range [100,199] is that it, or is there more?
3686 \@tmpstrctr=#1\relax \@modulo{\@tmpstrctr}{100}%
3687 \ifnum \@tmpstrctr=0\relax
3688 \eappto#2{\@hundredth}%
3689 \else
3690 \ifnum#1>1000\relax
3691 \appto#2{einhundert}%
3692 \else
3693 \eappto#2{einhundert}%
3694 \fi
3695 \fi
3696 \fi
3697 \fi
3698 \@strctr=#1\relax
3699 \@modulo{\@strctr}{100}%
3700 \ifthenelse{\@strctr=0 \and #1>0}{-}{%
3701 \@numberunderhundredthgerman{\@strctr}{#2}%
3702 }%
3703 }

    Load fc-germanb.def if not already loaded
3704 \FCloadlang{germanb}

```

9.4.8 fc-germanb.def

```

3705 \ProvidesFCLanguage{germanb}[2012/06/18]

    Load fc-german.def if not already loaded
3706 \FCloadlang{german}

    Set germanb to be equivalent to german.
3707 \let \@ordinalMgermanb=\@ordinalMgerman
3708 \let \@ordinalFgermanb=\@ordinalFgerman
3709 \let \@ordinalNgermanb=\@ordinalNgerman
3710 \let \@numberstringMgermanb=\@numberstringMgerman

```

```

3711 \let\@numberstringFgermanb=\@numberstringFgerman
3712 \let\@numberstringNgermanb=\@numberstringNgerman
3713 \let\@NumberstringMgermanb=\@NumberstringMgerman
3714 \let\@NumberstringFgermanb=\@NumberstringFgerman
3715 \let\@NumberstringNgermanb=\@NumberstringNgerman
3716 \let\@ordinalstringMgermanb=\@ordinalstringMgerman
3717 \let\@ordinalstringFgermanb=\@ordinalstringFgerman
3718 \let\@ordinalstringNgermanb=\@ordinalstringNgerman
3719 \let\@OrdinalstringMgermanb=\@OrdinalstringMgerman
3720 \let\@OrdinalstringFgermanb=\@OrdinalstringFgerman
3721 \let\@OrdinalstringNgermanb=\@OrdinalstringNgerman

```

9.4.9 fc-italian

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

```

3722 \ProvidesFCLanguage{italian}[2012/06/18]
3723
3724 \RequirePackage{itnumpar}
3725
3726 \newcommand{\@numberstringMitalian}[2]{%
3727   \edef#2{\noexpand\printnumeroinparole{#1}}%
3728 }
3729
3730 \newcommand{\@numberstringFitalian}[2]{%
3731   \edef#2{\noexpand\printnumeroinparole{#1}}%
3732 }
3733 \newcommand{\@NumberstringMitalian}[2]{%
3734   \edef#2{\noexpand\printNumeroinparole{#1}}%
3735 }
3736 \newcommand{\@NumberstringFitalian}[2]{%
3737   \edef#2{\noexpand\printNumeroinparole{#1}}%
3738 }
3739 \newcommand{\@ordinalstringMitalian}[2]{%
3740   \edef#2{\noexpand\printordinalem{#1}}%
3741 }
3742 \newcommand{\@ordinalstringFitalian}[2]{%
3743   \edef#2{\noexpand\printordinalef{#1}}%
3744 }
3745 \newcommand{\@OrdinalstringMitalian}[2]{%
3746   \edef#2{\noexpand\printOrdinalem{#1}}%
3747 }
3748 \newcommand{\@OrdinalstringFitalian}[2]{%
3749   \edef#2{\noexpand\printOrdinalef{#1}}%
3750 }
3751 \newcommand{\@ordinalMitalian}[2]{%
3752   \edef#2{#1\relax\noexpand\fmtord{o}}%
3753 }
3754 \newcommand{\@ordinalFitalian}[2]{%
3755   \edef#2{#1\relax\noexpand\fmtord{a}}%

```

9.4.10 fc-ngerman.def

```
3755 \ProvidesFCLanguage{ngerman}[2012/06/18]
3756 \FCloadlang{german}
3757 \FCloadlang{ngermanb}
```

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

```
3758 \let\@ordinalMngerman=\@ordinalMgerman
3759 \let\@ordinalFngerman=\@ordinalFgerman
3760 \let\@ordinalNngerman=\@ordinalNgerman
3761 \let\@numberstringMngerman=\@numberstringMgerman
3762 \let\@numberstringFngerman=\@numberstringFgerman
3763 \let\@numberstringNngerman=\@numberstringNgerman
3764 \let\@NumberstringMngerman=\@NumberstringMgerman
3765 \let\@NumberstringFngerman=\@NumberstringFgerman
3766 \let\@NumberstringNngerman=\@NumberstringNgerman
3767 \let\@ordinalstringMngerman=\@ordinalstringMgerman
3768 \let\@ordinalstringFngerman=\@ordinalstringFgerman
3769 \let\@ordinalstringNngerman=\@ordinalstringNgerman
3770 \let\@OrdinalstringMngerman=\@OrdinalstringMgerman
3771 \let\@OrdinalstringFngerman=\@OrdinalstringFgerman
3772 \let\@OrdinalstringNngerman=\@OrdinalstringNgerman
```

9.4.11 fc-ngermanb.def

```
3773 \ProvidesFCLanguage{ngermanb}[2012/06/18]
3774 \FCloadlang{german}
```

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

```
3775 \let\@ordinalMngermanb=\@ordinalMgerman
3776 \let\@ordinalFngermanb=\@ordinalFgerman
3777 \let\@ordinalNngermanb=\@ordinalNgerman
3778 \let\@numberstringMngermanb=\@numberstringMgerman
3779 \let\@numberstringFngermanb=\@numberstringFgerman
3780 \let\@numberstringNngermanb=\@numberstringNgerman
3781 \let\@NumberstringMngermanb=\@NumberstringMgerman
3782 \let\@NumberstringFngermanb=\@NumberstringFgerman
3783 \let\@NumberstringNngermanb=\@NumberstringNgerman
3784 \let\@ordinalstringMngermanb=\@ordinalstringMgerman
3785 \let\@ordinalstringFngermanb=\@ordinalstringFgerman
3786 \let\@ordinalstringNngermanb=\@ordinalstringNgerman
3787 \let\@OrdinalstringMngermanb=\@OrdinalstringMgerman
3788 \let\@OrdinalstringFngermanb=\@OrdinalstringFgerman
3789 \let\@OrdinalstringNngermanb=\@OrdinalstringNgerman
```

Load fc-ngerman.def if not already loaded

```
3790 \FCloadlang{ngerman}
```

9.4.12 fc-portuges.def

Portuguse definitions

3791 \ProvidesFCLanguage{portuges}[2012/06/18]

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

```
3792 \newcommand*{\@ordinalMportuges}[2]{%
3793 \ifnum#1=0\relax
3794   \edef#2{\number#1}%
3795 \else
3796   \edef#2{\number#1\relax\noexpand\fmtord{o}}%
3797 \fi}
```

Feminine:

```
3798 \newcommand*{\@ordinalFportuges}[2]{%
3799 \ifnum#1=0\relax
3800   \edef#2{\number#1}%
3801 \else
3802   \edef#2{\number#1\relax\noexpand\fmtord{a}}%
3803 \fi}
```

Make neuter same as masculine:

```
3804 \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):

```
3805 \newcommand*{\@unitstringportuges}[1]{%
3806 \ifcase#1\relax
3807 zero%
3808 \or um%
3809 \or dois%
3810 \or tr\~es%
3811 \or quatro%
3812 \or cinco%
3813 \or seis%
3814 \or sete%
3815 \or oito%
3816 \or nove%
3817 \fi
3818 }
3819 % \end{macrocode}
3820 % As above, but for feminine:
3821 % \begin{macrocode}
3822 \newcommand*{\@unitstringFportuges}[1]{%
3823 \ifcase#1\relax
3824 zero%
3825 \or uma%
3826 \or duas%
3827 \or tr\~es%
3828 \or quatro%
```

```

3829 \or cinco%
3830 \or seis%
3831 \or sete%
3832 \or oito%
3833 \or nove%
3834 \fi
3835 }

```

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

```

3836 \newcommand*{\@@tenstringportuges}[1]{%
3837 \ifcase#1\relax
3838 \or dez%
3839 \or vinte%
3840 \or trinta%
3841 \or quarenta%
3842 \or cinq\ "uenta%
3843 \or sessenta%
3844 \or setenta%
3845 \or oitenta%
3846 \or noventa%
3847 \or cem%
3848 \fi
3849 }

```

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

```

3850 \newcommand*{\@@teenstringportuges}[1]{%
3851 \ifcase#1\relax
3852 dez%
3853 \or onze%
3854 \or doze%
3855 \or treze%
3856 \or quatorze%
3857 \or quinze%
3858 \or dezesesseis%
3859 \or dezessete%
3860 \or dezoito%
3861 \or dezenove%
3862 \fi
3863 }

```

Hundreds:

```

3864 \newcommand*{\@@hundredstringportuges}[1]{%
3865 \ifcase#1\relax
3866 \or cento%
3867 \or duzentos%
3868 \or trezentos%
3869 \or quatrocentos%
3870 \or quinhentos%
3871 \or seiscentos%
3872 \or setecentos%
3873 \or oitocentos%

```

3874 \or novecentos%
3875 \fi}

Hundreds (feminine):

3876 \newcommand*{\@@hundredstringFportuges}[1]{%
3877 \ifcase#1\relax
3878 \or cento%
3879 \or duzentas%
3880 \or trezentas%
3881 \or quatrocentas%
3882 \or quinhentas%
3883 \or seiscentas%
3884 \or setecentas%
3885 \or oitocentas%
3886 \or novecentas%
3887 \fi}

Units (initial letter in upper case):

3888 \newcommand*{\@@Unitstringportuges}[1]{%
3889 \ifcase#1\relax
3890 Zero%
3891 \or Um%
3892 \or Dois%
3893 \or Tr\~es%
3894 \or Quatro%
3895 \or Cinco%
3896 \or Seis%
3897 \or Sete%
3898 \or Oito%
3899 \or Nove%
3900 \fi
3901 }

As above, but feminine:

3902 \newcommand*{\@@UnitstringFportuges}[1]{%
3903 \ifcase#1\relax
3904 Zera%
3905 \or Uma%
3906 \or Duas%
3907 \or Tr\~es%
3908 \or Quatro%
3909 \or Cinco%
3910 \or Seis%
3911 \or Sete%
3912 \or Oito%
3913 \or Nove%
3914 \fi
3915 }

Tens (with initial letter in upper case):

3916 \newcommand*{\@@Tenstringportuges}[1]{%

```

3917 \ifcase#1\relax
3918 \or Dez%
3919 \or Vinte%
3920 \or Trinta%
3921 \or Quarenta%
3922 \or Cinq\"uenta%
3923 \or Sessenta%
3924 \or Setenta%
3925 \or Oitenta%
3926 \or Noventa%
3927 \or Cem%
3928 \fi
3929 }

```

Teens (with initial letter in upper case):

```

3930 \newcommand*{\@@Teenstringportuges}[1]{%
3931 \ifcase#1\relax
3932 Dez%
3933 \or Onze%
3934 \or Doze%
3935 \or Treze%
3936 \or Quatorze%
3937 \or Quinze%
3938 \or Dezesesseis%
3939 \or Dezesesete%
3940 \or Dezoito%
3941 \or Dezenove%
3942 \fi
3943 }

```

Hundreds (with initial letter in upper case):

```

3944 \newcommand*{\@@Hundredstringportuges}[1]{%
3945 \ifcase#1\relax
3946 \or Cento%
3947 \or Duzentos%
3948 \or Trezentos%
3949 \or Quatrocentos%
3950 \or Quinhentos%
3951 \or Seiscentos%
3952 \or Setecentos%
3953 \or Oitocentos%
3954 \or Novecentos%
3955 \fi}

```

As above, but feminine:

```

3956 \newcommand*{\@@HundredstringFportuges}[1]{%
3957 \ifcase#1\relax
3958 \or Cento%
3959 \or Duzentas%
3960 \or Trezentas%
3961 \or Quatrocentas%

```

```

3962 \or Quinhentas%
3963 \or Seiscentas%
3964 \or Setecentas%
3965 \or Oitocentas%
3966 \or Novecentas%
3967 \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.)

```

3968 \DeclareRobustCommand{\@numberstringMportuges}[2]{%
3969 \let\@unitstring=\@unitstringportuges
3970 \let\@teenstring=\@teenstringportuges
3971 \let\@tenstring=\@tenstringportuges
3972 \let\@hundredstring=\@hundredstringportuges
3973 \def\@hundred{cem}\def\@thousand{mil}%
3974 \def\@andname{e}%
3975 \@numberstringportuges{#1}{#2}}

```

As above, but feminine form:

```

3976 \DeclareRobustCommand{\@numberstringFportuges}[2]{%
3977 \let\@unitstring=\@unitstringFportuges
3978 \let\@teenstring=\@teenstringportuges
3979 \let\@tenstring=\@tenstringportuges
3980 \let\@hundredstring=\@hundredstringFportuges
3981 \def\@hundred{cem}\def\@thousand{mil}%
3982 \def\@andname{e}%
3983 \@numberstringportuges{#1}{#2}}

```

Make neuter same as masculine:

```

3984 \let\@numberstringNportuges\@numberstringMportuges

```

As above, but initial letters in upper case:

```

3985 \DeclareRobustCommand{\@NumberstringMportuges}[2]{%
3986 \let\@unitstring=\@Unitstringportuges
3987 \let\@teenstring=\@Teenstringportuges
3988 \let\@tenstring=\@Tenstringportuges
3989 \let\@hundredstring=\@Hundredstringportuges
3990 \def\@hundred{Cem}\def\@thousand{Mil}%
3991 \def\@andname{e}%
3992 \@numberstringportuges{#1}{#2}}

```

As above, but feminine form:

```

3993 \DeclareRobustCommand{\@NumberstringFportuges}[2]{%
3994 \let\@unitstring=\@UnitstringFportuges
3995 \let\@teenstring=\@Teenstringportuges
3996 \let\@tenstring=\@Tenstringportuges
3997 \let\@hundredstring=\@HundredstringFportuges
3998 \def\@hundred{Cem}\def\@thousand{Mil}%
3999 \def\@andname{e}%

```


4000 \@@numberstringportuges{#1}{#2}}

Make neuter same as masculine:

4001 \let\@NumberstringNportuges\@NumberstringMportuges

As above, but for ordinals.

4002 \DeclareRobustCommand{\@ordinalstringMportuges}[2]{%

4003 \let\@unitthstring=\@unitthstringportuges

4004 \let\@unitstring=\@unitstringportuges

4005 \let\@teenthstring=\@teenthstringportuges

4006 \let\@tenthstring=\@tenthstringportuges

4007 \let\@hundredthstring=\@hundredthstringportuges

4008 \def\@thousandth{mil\'esimo}%

4009 \@@ordinalstringportuges{#1}{#2}}

Feminine form:

4010 \DeclareRobustCommand{\@ordinalstringFportuges}[2]{%

4011 \let\@unitthstring=\@unitthstringFportuges

4012 \let\@unitstring=\@unitstringFportuges

4013 \let\@teenthstring=\@teenthstringFportuges

4014 \let\@tenthstring=\@tenthstringFportuges

4015 \let\@hundredthstring=\@hundredthstringFportuges

4016 \def\@thousandth{mil\'esima}%

4017 \@@ordinalstringportuges{#1}{#2}}

Make neuter same as masculine:

4018 \let\@ordinalstringNportuges\@ordinalstringMportuges

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

4019 \DeclareRobustCommand{\@OrdinalstringMportuges}[2]{%

4020 \let\@unitthstring=\@Unitthstringportuges

4021 \let\@unitstring=\@Unitstringportuges

4022 \let\@teenthstring=\@Teenthstringportuges

4023 \let\@tenthstring=\@Tenthstringportuges

4024 \let\@hundredthstring=\@Hundredthstringportuges

4025 \def\@thousandth{Mil\'esimo}%

4026 \@@ordinalstringportuges{#1}{#2}}

Feminine form:

4027 \DeclareRobustCommand{\@OrdinalstringFportuges}[2]{%

4028 \let\@unitthstring=\@UnitthstringFportuges

4029 \let\@unitstring=\@UnitstringFportuges

4030 \let\@teenthstring=\@TeenthstringFportuges

4031 \let\@tenthstring=\@TenthstringFportuges

4032 \let\@hundredthstring=\@HundredthstringFportuges

4033 \def\@thousandth{Mil\'esima}%

4034 \@@ordinalstringportuges{#1}{#2}}

Make neuter same as masculine:

4035 \let\@OrdinalstringNportuges\@OrdinalstringMportuges

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

```

4036 \newcommand*{\@@unitthstringportuges}[1]{%
4037 \ifcase#1\relax
4038 zero%
4039 \or primeiro%
4040 \or segundo%
4041 \or terceiro%
4042 \or quarto%
4043 \or quinto%
4044 \or sexto%
4045 \or s\'etimo%
4046 \or oitavo%
4047 \or nono%
4048 \fi
4049 }

```

Tens:

```

4050 \newcommand*{\@@tenthstringportuges}[1]{%
4051 \ifcase#1\relax
4052 \or d\'ecimo%
4053 \or vig\'esimo%
4054 \or trig\'esimo%
4055 \or quadrag\'esimo%
4056 \or q\'uinquag\'esimo%
4057 \or sexag\'esimo%
4058 \or setuag\'esimo%
4059 \or octog\'esimo%
4060 \or nonag\'esimo%
4061 \fi
4062 }

```

Teens:

```

4063 \newcommand*{\@@teenthstringportuges}[1]{%
4064 \@tenthstring{1}%
4065 \ifnum#1>0\relax
4066 -\@unitthstring{#1}%
4067 \fi}

```

Hundreds:

```

4068 \newcommand*{\@@hundredthstringportuges}[1]{%
4069 \ifcase#1\relax
4070 \or cent\'esimo%
4071 \or ducent\'esimo%
4072 \or trecent\'esimo%
4073 \or quadringent\'esimo%
4074 \or q\'uingent\'esimo%
4075 \or seiscent\'esimo%
4076 \or setingent\'esimo%
4077 \or octingent\'esimo%
4078 \or nongent\'esimo%
4079 \fi}

```

Units (feminine):

```
4080 \newcommand*{\@@unitthstringFportuges}[1]{%
4081 \ifcase#1\relax
4082 zero%
4083 \or primeira%
4084 \or segunda%
4085 \or terceira%
4086 \or quarta%
4087 \or quinta%
4088 \or sexta%
4089 \or s\'etima%
4090 \or oitava%
4091 \or nona%
4092 \fi
4093 }
```

Tens (feminine):

```
4094 \newcommand*{\@@tenthstringFportuges}[1]{%
4095 \ifcase#1\relax
4096 \or d\'ecima%
4097 \or vig\'esima%
4098 \or trig\'esima%
4099 \or quadrag\'esima%
4100 \or q\'uinquag\'esima%
4101 \or sexag\'esima%
4102 \or setuag\'esima%
4103 \or octog\'esima%
4104 \or nonag\'esima%
4105 \fi
4106 }
```

Hundreds (feminine):

```
4107 \newcommand*{\@@hundredthstringFportuges}[1]{%
4108 \ifcase#1\relax
4109 \or cent\'esima%
4110 \or ducent\'esima%
4111 \or trecent\'esima%
4112 \or quadringent\'esima%
4113 \or q\'uingent\'esima%
4114 \or seiscent\'esima%
4115 \or setingent\'esima%
4116 \or octingent\'esima%
4117 \or nongent\'esima%
4118 \fi}
```

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

```
4119 \newcommand*{\@@Unitthstringportuges}[1]{%
4120 \ifcase#1\relax
4121 Zero%
4122 \or Primeiro%
```

```

4123 \or Segundo%
4124 \or Terceiro%
4125 \or Quarto%
4126 \or Quinto%
4127 \or Sexto%
4128 \or S\'etimo%
4129 \or Oitavo%
4130 \or Nono%
4131 \fi
4132 }

```

Tens:

```

4133 \newcommand*{\@@Tenthstringportuges}[1]{%
4134 \ifcase#1\relax
4135 \or D\'ecimo%
4136 \or Vig\'esimo%
4137 \or Trig\'esimo%
4138 \or Quadrag\'esimo%
4139 \or Q\'uinquag\'esimo%
4140 \or Sexag\'esimo%
4141 \or Setuag\'esimo%
4142 \or Octog\'esimo%
4143 \or Nonag\'esimo%
4144 \fi
4145 }

```

Hundreds:

```

4146 \newcommand*{\@@Hundredthstringportuges}[1]{%
4147 \ifcase#1\relax
4148 \or Cent\'esimo%
4149 \or Ducent\'esimo%
4150 \or Trecent\'esimo%
4151 \or Quadringent\'esimo%
4152 \or Q\'uingent\'esimo%
4153 \or Seiscent\'esimo%
4154 \or Setingent\'esimo%
4155 \or Octingent\'esimo%
4156 \or Nongent\'esimo%
4157 \fi}

```

As above, but feminine. Units:

```

4158 \newcommand*{\@@UnitthstringFportuges}[1]{%
4159 \ifcase#1\relax
4160 Zera%
4161 \or Primeira%
4162 \or Segunda%
4163 \or Terceira%
4164 \or Quarta%
4165 \or Quinta%
4166 \or Sexta%
4167 \or S\'etima%

```

```

4168 \or Oitava%
4169 \or Nona%
4170 \fi
4171 }

```

Tens (feminine);

```

4172 \newcommand*{\@@TenthstringFportuges}[1]{%
4173 \ifcase#1\relax
4174 \or D\'ecima%
4175 \or Vig\'esima%
4176 \or Trig\'esima%
4177 \or Quadrag\'esima%
4178 \or Q\'uinquag\'esima%
4179 \or Sexag\'esima%
4180 \or Setuag\'esima%
4181 \or Octog\'esima%
4182 \or Nonag\'esima%
4183 \fi
4184 }

```

Hundreds (feminine):

```

4185 \newcommand*{\@@HundredthstringFportuges}[1]{%
4186 \ifcase#1\relax
4187 \or Cent\'esima%
4188 \or Ducent\'esima%
4189 \or Trecent\'esima%
4190 \or Quadringent\'esima%
4191 \or Q\'uingent\'esima%
4192 \or Seiscent\'esima%
4193 \or Setingent\'esima%
4194 \or Octingent\'esima%
4195 \or Nongent\'esima%
4196 \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.)

```

4197 \newcommand*{\@@numberstringportuges}[2]{%
4198 \ifnum#1>99999
4199 \PackageError{fmtcount}{Out of range}%
4200 {This macro only works for values less than 100000}%
4201 \else
4202 \ifnum#1<0
4203 \PackageError{fmtcount}{Negative numbers not permitted}%
4204 {This macro does not work for negative numbers, however
4205 you can try typing "minus" first, and then pass the modulus of
4206 this number}%
4207 \fi
4208 \fi

```

```

4209 \def#2{}%
4210 \@strctr=#1\relax \divide\@strctr by 1000\relax
4211 \ifnum\@strctr>9
    #1 is greater or equal to 10000
4212 \divide\@strctr by 10
4213 \ifnum\@strctr>1\relax
4214 \let\@@fc@numstr#2\relax
4215 \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
4216 \@strctr=#1 \divide\@strctr by 1000\relax
4217 \@modulo{\@strctr}{10}%
4218 \ifnum\@strctr>0
4219 \ifnum\@strctr=1\relax
4220 \let\@@fc@numstr#2\relax
4221 \edef#2{\@@fc@numstr\ \@andname}%
4222 \fi
4223 \let\@@fc@numstr#2\relax
4224 \edef#2{\@@fc@numstr\ \@unitstring{\@strctr}}%
4225 \fi
4226 \else
4227 \@strctr=#1\relax
4228 \divide\@strctr by 1000\relax
4229 \@modulo{\@strctr}{10}%
4230 \let\@@fc@numstr#2\relax
4231 \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
4232 \fi
4233 \let\@@fc@numstr#2\relax
4234 \edef#2{\@@fc@numstr\ \@thousand}%
4235 \else
4236 \ifnum\@strctr>0\relax
4237 \ifnum\@strctr>1\relax
4238 \let\@@fc@numstr#2\relax
4239 \edef#2{\@@fc@numstr\@unitstring{\@strctr}\ }%
4240 \fi
4241 \let\@@fc@numstr#2\relax
4242 \edef#2{\@@fc@numstr\@thousand}%
4243 \fi
4244 \fi
4245 \@strctr=#1\relax \@modulo{\@strctr}{1000}%
4246 \divide\@strctr by 100\relax
4247 \ifnum\@strctr>0\relax
4248 \ifnum#1>1000 \relax
4249 \let\@@fc@numstr#2\relax
4250 \edef#2{\@@fc@numstr\ }%
4251 \fi
4252 \@tmpstrctr=#1\relax
4253 \@modulo{\@tmpstrctr}{1000}%
4254 \let\@@fc@numstr#2\relax
4255 \ifnum\@tmpstrctr=100\relax
4256 \edef#2{\@@fc@numstr\@tenstring{10}}%

```

```

4257 \else
4258 \edef#2{\@@fc@numstr\@hundredstring{\@strctr}}%
4259 \fi%
4260 \fi
4261 \@strctr=#1\relax \@modulo{\@strctr}{100}%
4262 \ifnum#1>100\relax
4263 \ifnum\@strctr>0\relax
4264 \let\@@fc@numstr#2\relax
4265 \edef#2{\@@fc@numstr\ \@andname\ }%
4266 \fi
4267 \fi
4268 \ifnum\@strctr>19\relax
4269 \divide\@strctr by 10\relax
4270 \let\@@fc@numstr#2\relax
4271 \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
4272 \@strctr=#1\relax \@modulo{\@strctr}{10}%
4273 \ifnum\@strctr>0
4274 \ifnum\@strctr=1\relax
4275 \let\@@fc@numstr#2\relax
4276 \edef#2{\@@fc@numstr\ \@andname}%
4277 \else
4278 \ifnum#1>100\relax
4279 \let\@@fc@numstr#2\relax
4280 \edef#2{\@@fc@numstr\ \@andname}%
4281 \fi
4282 \fi
4283 \let\@@fc@numstr#2\relax
4284 \edef#2{\@@fc@numstr\ \@unitstring{\@strctr}}%
4285 \fi
4286 \else
4287 \ifnum\@strctr<10\relax
4288 \ifnum\@strctr=0\relax
4289 \ifnum#1<100\relax
4290 \let\@@fc@numstr#2\relax
4291 \edef#2{\@@fc@numstr\@unitstring{\@strctr}}%
4292 \fi
4293 \else%(>0,<10)
4294 \let\@@fc@numstr#2\relax
4295 \edef#2{\@@fc@numstr\@unitstring{\@strctr}}%
4296 \fi
4297 \else%>10
4298 \@modulo{\@strctr}{10}%
4299 \let\@@fc@numstr#2\relax
4300 \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
4301 \fi
4302 \fi
4303 }

```

As above, but for ordinals.

```

4304 \newcommand*{\@@ordinalstringportuges}[2]{%

```

```

4305 \@strctr=#1\relax
4306 \ifnum#1>99999
4307 \PackageError{fmtcount}{Out of range}%
4308 {This macro only works for values less than 100000}%
4309 \else
4310 \ifnum#1<0
4311 \PackageError{fmtcount}{Negative numbers not permitted}%
4312 {This macro does not work for negative numbers, however
4313 you can try typing "minus" first, and then pass the modulus of
4314 this number}%
4315 \else
4316 \def#2{}%
4317 \ifnum\@strctr>999\relax
4318   \divide\@strctr by 1000\relax
4319   \ifnum\@strctr>1\relax
4320     \ifnum\@strctr>9\relax
4321       \@tmpstrctr=\@strctr
4322       \ifnum\@strctr<20
4323         \@modulo{\@tmpstrctr}{10}%
4324         \let\@@fc@ordstr#2\relax
4325         \edef#2{\@@fc@ordstr\@teenthstring{\@tmpstrctr}}%
4326       \else
4327         \divide\@tmpstrctr by 10\relax
4328         \let\@@fc@ordstr#2\relax
4329         \edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
4330         \@tmpstrctr=\@strctr
4331         \@modulo{\@tmpstrctr}{10}%
4332         \ifnum\@tmpstrctr>0\relax
4333           \let\@@fc@ordstr#2\relax
4334           \edef#2{\@@fc@ordstr\@unitthstring{\@tmpstrctr}}%
4335         \fi
4336       \fi
4337     \else
4338       \let\@@fc@ordstr#2\relax
4339       \edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
4340     \fi
4341   \fi
4342   \let\@@fc@ordstr#2\relax
4343   \edef#2{\@@fc@ordstr\@thousandth}%
4344 \fi
4345 \@strctr=#1\relax
4346 \@modulo{\@strctr}{1000}%
4347 \ifnum\@strctr>99\relax
4348   \@tmpstrctr=\@strctr
4349   \divide\@tmpstrctr by 100\relax
4350   \ifnum#1>1000\relax
4351     \let\@@fc@ordstr#2\relax
4352     \edef#2{\@@fc@ordstr-}%
4353   \fi

```



```

4354 \let\@@fc@ordstr#2\relax
4355 \edef#2{\@@fc@ordstr\@hundredthstring{\@tmpstrctr}}%
4356 \fi
4357 \@modulo{\@strctr}{100}%
4358 \ifnum#1>99\relax
4359 \ifnum\@strctr>0\relax
4360 \let\@@fc@ordstr#2\relax
4361 \edef#2{\@@fc@ordstr-}%
4362 \fi
4363 \fi
4364 \ifnum\@strctr>9\relax
4365 \@tmpstrctr=\@strctr
4366 \divide\@tmpstrctr by 10\relax
4367 \let\@@fc@ordstr#2\relax
4368 \edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
4369 \@tmpstrctr=\@strctr
4370 \@modulo{\@tmpstrctr}{10}%
4371 \ifnum\@tmpstrctr>0\relax
4372 \let\@@fc@ordstr#2\relax
4373 \edef#2{\@@fc@ordstr-\@unitthstring{\@tmpstrctr}}%
4374 \fi
4375 \else
4376 \ifnum\@strctr=0\relax
4377 \ifnum#1=0\relax
4378 \let\@@fc@ordstr#2\relax
4379 \edef#2{\@@fc@ordstr\@unitstring{0}}%
4380 \fi
4381 \else
4382 \let\@@fc@ordstr#2\relax
4383 \edef#2{\@@fc@ordstr\@unitthstring{\@strctr}}%
4384 \fi
4385 \fi
4386 \fi
4387 \fi
4388 }

```

9.4.13 fc-spanish.def

Spanish definitions

```
4389 \ProvidesFCLanguage{spanish}[2012/06/18]
```

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

```

4390 \newcommand{\@ordinalMspanish}[2]{%
4391 \edef#2{\number#1\relax\noexpand\fmtord{o}}}

```

Feminine:

```

4392 \newcommand{\@ordinalFspanish}[2]{%
4393 \edef#2{\number#1\relax\noexpand\fmtord{a}}}

```

Make neuter same as masculine:

```
4394 \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):

```
4395 \newcommand{\@@unitstringspanish}[1]{%
4396 \ifcase#1\relax
4397 cero%
4398 \or uno%
4399 \or dos%
4400 \or tres%
4401 \or cuatro%
4402 \or cinco%
4403 \or seis%
4404 \or siete%
4405 \or ocho%
4406 \or nueve%
4407 \fi
4408 }
```

Feminine:

```
4409 \newcommand{\@@unitstringFspanish}[1]{%
4410 \ifcase#1\relax
4411 cera%
4412 \or una%
4413 \or dos%
4414 \or tres%
4415 \or cuatro%
4416 \or cinco%
4417 \or seis%
4418 \or siete%
4419 \or ocho%
4420 \or nueve%
4421 \fi
4422 }
```

Tens (argument must go from 1 to 10):

```
4423 \newcommand{\@@tenstringspanish}[1]{%
4424 \ifcase#1\relax
4425 \or diez%
4426 \or veinte%
4427 \or treinta%
4428 \or cuarenta%
4429 \or cincuenta%
4430 \or sesenta%
4431 \or setenta%
4432 \or ochenta%
4433 \or noventa%
4434 \or cien%
```

4435 \fi

4436 }

Teens:

4437 \newcommand{\@@teenstringspanish}[1]{%

4438 \ifcase#1\relax

4439 diez%

4440 \or once%

4441 \or doce%

4442 \or trece%

4443 \or catorce%

4444 \or quince%

4445 \or diecis\'eis%

4446 \or diecisiete%

4447 \or dieciocho%

4448 \or diecinueve%

4449 \fi

4450 }

Twenties:

4451 \newcommand{\@@twentystringspanish}[1]{%

4452 \ifcase#1\relax

4453 veinte%

4454 \or veintiuno%

4455 \or veintid\'os%

4456 \or veintitr\'es%

4457 \or veinticuatro%

4458 \or veinticinco%

4459 \or veintis\'eis%

4460 \or veintisiete%

4461 \or veintiocho%

4462 \or veintinueve%

4463 \fi}

Feminine form:

4464 \newcommand{\@@twentystringFspanish}[1]{%

4465 \ifcase#1\relax

4466 veinte%

4467 \or veintiuna%

4468 \or veintid\'os%

4469 \or veintitr\'es%

4470 \or veinticuatro%

4471 \or veinticinco%

4472 \or veintis\'eis%

4473 \or veintisiete%

4474 \or veintiocho%

4475 \or veintinueve%

4476 \fi}

Hundreds:

4477 \newcommand{\@@hundredstringspanish}[1]{%

```

4478 \ifcase#1\relax
4479 \or ciento%
4480 \or doscientos%
4481 \or trescientos%
4482 \or cuatrocientos%
4483 \or quinientos%
4484 \or seiscientos%
4485 \or setecientos%
4486 \or ochocientos%
4487 \or novecientos%
4488 \fi}

```

Feminine form:

```

4489 \newcommand{\@@hundredstringFspanish}[1]{%
4490 \ifcase#1\relax
4491 \or cienta%
4492 \or doscientas%
4493 \or trescientas%
4494 \or cuatrocientas%
4495 \or quinientas%
4496 \or seiscientas%
4497 \or setecientas%
4498 \or ochocientas%
4499 \or novecientas%
4500 \fi}

```

As above, but with initial letter uppercase:

```

4501 \newcommand{\@@Unitstringspanish}[1]{%
4502 \ifcase#1\relax
4503 Cero%
4504 \or Uno%
4505 \or Dos%
4506 \or Tres%
4507 \or Cuatro%
4508 \or Cinco%
4509 \or Seis%
4510 \or Siete%
4511 \or Ocho%
4512 \or Nueve%
4513 \fi
4514 }

```

Feminine form:

```

4515 \newcommand{\@@UnitstringFspanish}[1]{%
4516 \ifcase#1\relax
4517 Cera%
4518 \or Una%
4519 \or Dos%
4520 \or Tres%
4521 \or Cuatro%
4522 \or Cinco%

```

```

4523 \or Seis%
4524 \or Siete%
4525 \or Ocho%
4526 \or Nueve%
4527 \fi
4528 }

```

Tens:

```

4529 %\changes{2.0}{2012-06-18}{fixed spelling mistake (correction
4530 %provided by Fernando Maldonado)}
4531 \newcommand{\@@Tenstringspanish}[1]{%
4532 \ifcase#1\relax
4533 \or Diez%
4534 \or Veinte%
4535 \or Treinta%
4536 \or Cuarenta%
4537 \or Cincuenta%
4538 \or Sesenta%
4539 \or Setenta%
4540 \or Ochenta%
4541 \or Noventa%
4542 \or Cien%
4543 \fi
4544 }

```

Teens:

```

4545 \newcommand{\@@Teenstringspanish}[1]{%
4546 \ifcase#1\relax
4547 Diez%
4548 \or Once%
4549 \or Doce%
4550 \or Trece%
4551 \or Catorce%
4552 \or Quince%
4553 \or Diecis\'eis%
4554 \or Diecisiete%
4555 \or Dieciocho%
4556 \or Diecinueve%
4557 \fi
4558 }

```

Twenties:

```

4559 \newcommand{\@@Twentystringspanish}[1]{%
4560 \ifcase#1\relax
4561 Veinte%
4562 \or Veintiuno%
4563 \or Veintid\'os%
4564 \or Veintitr\'es%
4565 \or Veinticuatro%
4566 \or Veinticinco%
4567 \or Veintis\'eis%

```

```

4568 \or Veintisiete%
4569 \or Veintiocho%
4570 \or Veintinueve%
4571 \fi}

```

Feminine form:

```

4572 \newcommand{\@@TwentystringFspanish}[1]{%
4573 \ifcase#1\relax
4574 Veinte%
4575 \or Veintiuna%
4576 \or Veintid\'os%
4577 \or Veintitr\'es%
4578 \or Veinticuatro%
4579 \or Veinticinco%
4580 \or Veintis\'eis%
4581 \or Veintisiete%
4582 \or Veintiocho%
4583 \or Veintinueve%
4584 \fi}

```

Hundreds:

```

4585 \newcommand{\@@Hundredstringspanish}[1]{%
4586 \ifcase#1\relax
4587 \or Ciento%
4588 \or Doscientos%
4589 \or Trescientos%
4590 \or Cuatrocientos%
4591 \or Quinientos%
4592 \or Seiscientos%
4593 \or Setecientos%
4594 \or Ochocientos%
4595 \or Novecientos%
4596 \fi}

```

Feminine form:

```

4597 \newcommand{\@@HundredstringFspanish}[1]{%
4598 \ifcase#1\relax
4599 \or Cienta%
4600 \or Doscientas%
4601 \or Trescientas%
4602 \or Cuatrocientas%
4603 \or Quinientas%
4604 \or Seiscientas%
4605 \or Setecientas%
4606 \or Ochocientas%
4607 \or Novecientas%
4608 \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.)

```
4609 \DeclareRobustCommand{\@numberstringMspanish}[2]{%
4610 \let\@unitstring=\@unitstringspanish
4611 \let\@teenstring=\@teenstringspanish
4612 \let\@tenstring=\@tenstringspanish
4613 \let\@twentystring=\@twentystringspanish
4614 \let\@hundredstring=\@hundredstringspanish
4615 \def\@hundred{cien}\def\@thousand{mil}%
4616 \def\@andname{y}%
4617 \@numberstringspanish{#1}{#2}}
```

Feminine form:

```
4618 \DeclareRobustCommand{\@numberstringFspanish}[2]{%
4619 \let\@unitstring=\@unitstringFspanish
4620 \let\@teenstring=\@teenstringspanish
4621 \let\@tenstring=\@tenstringspanish
4622 \let\@twentystring=\@twentystringFspanish
4623 \let\@hundredstring=\@hundredstringFspanish
4624 \def\@hundred{cien}\def\@thousand{mil}%
4625 \def\@andname{b}%
4626 \@numberstringspanish{#1}{#2}}
```

Make neuter same as masculine:

```
4627 \let\@numberstringNspanish\@numberstringMspanish
```

As above, but initial letters in upper case:

```
4628 \DeclareRobustCommand{\@NumberstringMspanish}[2]{%
4629 \let\@unitstring=\@Unitstringspanish
4630 \let\@teenstring=\@Teenstringspanish
4631 \let\@tenstring=\@Tenstringspanish
4632 \let\@twentystring=\@Twentystringspanish
4633 \let\@hundredstring=\@Hundredstringspanish
4634 \def\@andname{y}%
4635 \def\@hundred{Cien}\def\@thousand{Mil}%
4636 \@numberstringspanish{#1}{#2}}
```

Feminine form:

```
4637 \DeclareRobustCommand{\@NumberstringFspanish}[2]{%
4638 \let\@unitstring=\@UnitstringFspanish
4639 \let\@teenstring=\@Teenstringspanish
4640 \let\@tenstring=\@Tenstringspanish
4641 \let\@twentystring=\@TwentystringFspanish
4642 \let\@hundredstring=\@HundredstringFspanish
4643 \def\@andname{b}%
4644 \def\@hundred{Cien}\def\@thousand{Mil}%
4645 \@numberstringspanish{#1}{#2}}
```

Make neuter same as masculine:

```
4646 \let\@NumberstringNspanish\@NumberstringMspanish
```

As above, but for ordinals.

```
4647 \DeclareRobustCommand{\@ordinalstringMspanish}[2]{%
4648 \let\@unitthstring=\@unitthstringspanish
4649 \let\@unitstring=\@unitstringspanish
4650 \let\@teenthstring=\@teenthstringspanish
4651 \let\@tenthstring=\@tenthstringspanish
4652 \let\@hundredthstring=\@hundredthstringspanish
4653 \def\@thousandth{mil\'esimo}%
4654 \@ordinalstringspanish{#1}{#2}}
```

Feminine form:

```
4655 \DeclareRobustCommand{\@ordinalstringFspanish}[2]{%
4656 \let\@unitthstring=\@unitthstringFspanish
4657 \let\@unitstring=\@unitstringFspanish
4658 \let\@teenthstring=\@teenthstringFspanish
4659 \let\@tenthstring=\@tenthstringFspanish
4660 \let\@hundredthstring=\@hundredthstringFspanish
4661 \def\@thousandth{mil\'esima}%
4662 \@ordinalstringspanish{#1}{#2}}
```

Make neuter same as masculine:

```
4663 \let\@ordinalstringNspanish\@ordinalstringMspanish
```

As above, but with initial letters in upper case.

```
4664 \DeclareRobustCommand{\@OrdinalstringMspanish}[2]{%
4665 \let\@unitthstring=\@Unitthstringspanish
4666 \let\@unitstring=\@Unitstringspanish
4667 \let\@teenthstring=\@Teenthstringspanish
4668 \let\@tenthstring=\@Tenthstringspanish
4669 \let\@hundredthstring=\@Hundredthstringspanish
4670 \def\@thousandth{Mil\'esimo}%
4671 \@Ordinalstringspanish{#1}{#2}}
```

Feminine form:

```
4672 \DeclareRobustCommand{\@OrdinalstringFspanish}[2]{%
4673 \let\@unitthstring=\@UnitthstringFspanish
4674 \let\@unitstring=\@UnitstringFspanish
4675 \let\@teenthstring=\@TeenthstringFspanish
4676 \let\@tenthstring=\@TenthstringFspanish
4677 \let\@hundredthstring=\@HundredthstringFspanish
4678 \def\@thousandth{Mil\'esima}%
4679 \@Ordinalstringspanish{#1}{#2}}
```

Make neuter same as masculine:

```
4680 \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:

```
4681 \newcommand{\@unitthstringspanish}[1]{%
4682 \ifcase#1\relax
4683 cero%
```



```

4684 \or primero%
4685 \or segundo%
4686 \or tercero%
4687 \or cuarto%
4688 \or quinto%
4689 \or sexto%
4690 \or s\'eptimo%
4691 \or octavo%
4692 \or noveno%
4693 \fi
4694 }

```

Tens:

```

4695 \newcommand{\@@tenthstringspanish}[1]{%
4696 \ifcase#1\relax
4697 \or d\'ecimo%
4698 \or vig\'esimo%
4699 \or trig\'esimo%
4700 \or cuadrag\'esimo%
4701 \or quincuag\'esimo%
4702 \or sexag\'esimo%
4703 \or septuag\'esimo%
4704 \or octog\'esimo%
4705 \or nonag\'esimo%
4706 \fi
4707 }

```

Teens:

```

4708 \newcommand{\@@teenthstringspanish}[1]{%
4709 \ifcase#1\relax
4710 d\'ecimo%
4711 \or und\'ecimo%
4712 \or duod\'ecimo%
4713 \or decimotercero%
4714 \or decimocuarto%
4715 \or decimoquinto%
4716 \or decimosexto%
4717 \or decimos\'eptimo%
4718 \or decimoctavo%
4719 \or decimonoveno%
4720 \fi
4721 }

```

Hundreds:

```

4722 \newcommand{\@@hundredthstringspanish}[1]{%
4723 \ifcase#1\relax
4724 \or cent\'esimo%
4725 \or ducent\'esimo%
4726 \or tricent\'esimo%
4727 \or cuadringent\'esimo%
4728 \or quingent\'esimo%

```

```

4729 \or sexcent\'esimo%
4730 \or septing\'esimo%
4731 \or octingent\'esimo%
4732 \or noningent\'esimo%
4733 \fi}

```

Units (feminine):

```

4734 \newcommand{\@@unitthstringFspanish}[1]{%
4735 \ifcase#1\relax
4736 cera%
4737 \or primera%
4738 \or segunda%
4739 \or tercera%
4740 \or cuarta%
4741 \or quinta%
4742 \or sexta%
4743 \or s\'eptima%
4744 \or octava%
4745 \or novena%
4746 \fi
4747 }

```

Tens (feminine):

```

4748 \newcommand{\@@tenthstringFspanish}[1]{%
4749 \ifcase#1\relax
4750 \or d\'ecima%
4751 \or vig\'esima%
4752 \or trig\'esima%
4753 \or cuadrag\'esima%
4754 \or quincuag\'esima%
4755 \or sexag\'esima%
4756 \or septuag\'esima%
4757 \or octog\'esima%
4758 \or nonag\'esima%
4759 \fi
4760 }

```

Teens (feminine)

```

4761 \newcommand{\@@teenthstringFspanish}[1]{%
4762 \ifcase#1\relax
4763 d\'ecima%
4764 \or und\'ecima%
4765 \or duod\'ecima%
4766 \or decimotercera%
4767 \or decimocuarta%
4768 \or decimoquinta%
4769 \or decimosexta%
4770 \or decimos\'eptima%
4771 \or decimoctava%
4772 \or decimonovena%
4773 \fi

```

4774 }

Hundreds (feminine)

```
4775 \newcommand{\@@hundredthstringFspanish}[1]{%
4776 \ifcase#1\relax
4777 \or cent\'esima%
4778 \or ducent\'esima%
4779 \or tricent\'esima%
4780 \or cuadringent\'esima%
4781 \or quingent\'esima%
4782 \or sexcent\'esima%
4783 \or septing\'esima%
4784 \or octingent\'esima%
4785 \or noningent\'esima%
4786 \fi}
```

As above, but with initial letters in upper case

```
4787 \newcommand{\@@Unitthstringspanish}[1]{%
4788 \ifcase#1\relax
4789 Cero%
4790 \or Primero%
4791 \or Segundo%
4792 \or Tercero%
4793 \or Cuarto%
4794 \or Quinto%
4795 \or Sexto%
4796 \or S\'eptimo%
4797 \or Octavo%
4798 \or Noveno%
4799 \fi
4800 }
```

Tens:

```
4801 \newcommand{\@@Tenthstringspanish}[1]{%
4802 \ifcase#1\relax
4803 \or D\'ecimo%
4804 \or Vig\'esimo%
4805 \or Trig\'esimo%
4806 \or Cuadrag\'esimo%
4807 \or Quincuag\'esimo%
4808 \or Sexag\'esimo%
4809 \or Septuag\'esimo%
4810 \or Octog\'esimo%
4811 \or Nonag\'esimo%
4812 \fi
4813 }
```

Teens:

```
4814 \newcommand{\@@Teenthstringspanish}[1]{%
4815 \ifcase#1\relax
4816 D\'ecimo%
```

```

4817 \or Und\'ecimo%
4818 \or Duod\'ecimo%
4819 \or Decimotercero%
4820 \or Decimocuarto%
4821 \or Decimoquinto%
4822 \or Decimosexto%
4823 \or Decimos\'eptimo%
4824 \or Decimoctavo%
4825 \or Decimonoveno%
4826 \fi
4827 }

```

Hundreds

```

4828 \newcommand{\@@Hundredthstringspanish}[1]{%
4829 \ifcase#1\relax
4830 \or Cent\'esimo%
4831 \or Ducent\'esimo%
4832 \or Tricent\'esimo%
4833 \or Cuadringent\'esimo%
4834 \or Quingent\'esimo%
4835 \or Sexcent\'esimo%
4836 \or Septing\'esimo%
4837 \or Octingent\'esimo%
4838 \or Noningent\'esimo%
4839 \fi}

```

As above, but feminine.

```

4840 \newcommand{\@@UnitthstringFspanish}[1]{%
4841 \ifcase#1\relax
4842 Cera%
4843 \or Primera%
4844 \or Segunda%
4845 \or Tercera%
4846 \or Cuarta%
4847 \or Quinta%
4848 \or Sexta%
4849 \or S\'eptima%
4850 \or Octava%
4851 \or Novena%
4852 \fi
4853 }

```

Tens (feminine)

```

4854 \newcommand{\@@TenthstringFspanish}[1]{%
4855 \ifcase#1\relax
4856 \or D\'esima%
4857 \or Vig\'esima%
4858 \or Trig\'esima%
4859 \or Cuadrag\'esima%
4860 \or Quincuag\'esima%
4861 \or Sexag\'esima%

```

```

4862 \or Septuag\'esima%
4863 \or Octog\'esima%
4864 \or Nonag\'esima%
4865 \fi
4866 }

```

Teens (feminine):

```

4867 \newcommand{\@@TeenthstringFspanish}[1]{%
4868 \ifcase#1\relax
4869 D\'esima%
4870 \or Und\'esima%
4871 \or Duod\'esima%
4872 \or Decimotercera%
4873 \or Decimocuarta%
4874 \or Decimoquinta%
4875 \or Decimosexta%
4876 \or Decimos\'eptima%
4877 \or Decimoctava%
4878 \or Decimonovena%
4879 \fi
4880 }

```

Hundreds (feminine):

```

4881 \newcommand{\@@HundredthstringFspanish}[1]{%
4882 \ifcase#1\relax
4883 \or Cent\'esima%
4884 \or Ducent\'esima%
4885 \or Tricent\'esima%
4886 \or Cuadringent\'esima%
4887 \or Quingent\'esima%
4888 \or Sexcent\'esima%
4889 \or Septing\'esima%
4890 \or Octingent\'esima%
4891 \or Noningent\'esima%
4892 \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.)

```

4893 \newcommand{\@@numberstringspanish}[2]{%
4894 \ifnum#1>99999
4895 \PackageError{fmtcount}{Out of range}%
4896 {This macro only works for values less than 100000}%
4897 \else
4898 \ifnum#1<0
4899 \PackageError{fmtcount}{Negative numbers not permitted}%
4900 {This macro does not work for negative numbers, however
4901 you can try typing "minus" first, and then pass the modulus of
4902 this number}%

```

```

4903 \fi
4904 \fi
4905 \def#2{}%
4906 \@strctr=#1\relax \divide\@strctr by 1000\relax
4907 \ifnum\@strctr>9

    #1 is greater or equal to 10000
4908 \divide\@strctr by 10
4909 \ifnum\@strctr>1
4910 \let\@@fc@numstr#2\relax
4911 \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
4912 \@strctr=#1 \divide\@strctr by 1000\relax
4913 \@modulo{\@strctr}{10}%
4914 \ifnum\@strctr>0\relax
4915 \let\@@fc@numstr#2\relax
4916 \edef#2{\@@fc@numstr\ \@andname\ \@unitstring{\@strctr}}%
4917 \fi
4918 \else
4919 \@strctr=#1\relax
4920 \divide\@strctr by 1000\relax
4921 \@modulo{\@strctr}{10}%
4922 \let\@@fc@numstr#2\relax
4923 \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
4924 \fi
4925 \let\@@fc@numstr#2\relax
4926 \edef#2{\@@fc@numstr\ \@thousand}%
4927 \else
4928 \ifnum\@strctr>0\relax
4929 \ifnum\@strctr>1\relax
4930 \let\@@fc@numstr#2\relax
4931 \edef#2{\@@fc@numstr\@unitstring{\@strctr}\ }%
4932 \fi
4933 \let\@@fc@numstr#2\relax
4934 \edef#2{\@@fc@numstr\@thousand}%
4935 \fi
4936 \fi
4937 \@strctr=#1\relax \@modulo{\@strctr}{1000}%
4938 \divide\@strctr by 100\relax
4939 \ifnum\@strctr>0\relax
4940 \ifnum#1>1000\relax
4941 \let\@@fc@numstr#2\relax
4942 \edef#2{\@@fc@numstr\ }%
4943 \fi
4944 \@tmpstrctr=#1\relax
4945 \@modulo{\@tmpstrctr}{1000}%
4946 \ifnum\@tmpstrctr=100\relax
4947 \let\@@fc@numstr#2\relax
4948 \edef#2{\@@fc@numstr\@tenstring{10}}%
4949 \else
4950 \let\@@fc@numstr#2\relax

```

```

4951 \edef#2{\@@fc@numstr\@hundredstring{\@strctr}}%
4952 \fi
4953 \fi
4954 \@strctr=#1\relax \@modulo{\@strctr}{100}%
4955 \ifnum#1>100\relax
4956 \ifnum\@strctr>0\relax
4957 \let\@@fc@numstr#2\relax

4958 \edef#2{\@@fc@numstr\ }%
4959 \fi
4960 \fi
4961 \ifnum\@strctr>29\relax
4962 \divide\@strctr by 10\relax
4963 \let\@@fc@numstr#2\relax
4964 \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
4965 \@strctr=#1\relax \@modulo{\@strctr}{10}%
4966 \ifnum\@strctr>0\relax
4967 \let\@@fc@numstr#2\relax
4968 \edef#2{\@@fc@numstr\ \@andname\ \@unitstring{\@strctr}}%
4969 \fi
4970 \else
4971 \ifnum\@strctr<10\relax
4972 \ifnum\@strctr=0\relax
4973 \ifnum#1<100\relax
4974 \let\@@fc@numstr#2\relax
4975 \edef#2{\@@fc@numstr\@unitstring{\@strctr}}%
4976 \fi
4977 \else
4978 \let\@@fc@numstr#2\relax
4979 \edef#2{\@@fc@numstr\@unitstring{\@strctr}}%
4980 \fi
4981 \else
4982 \ifnum\@strctr>19\relax
4983 \@modulo{\@strctr}{10}%
4984 \let\@@fc@numstr#2\relax
4985 \edef#2{\@@fc@numstr\@twentystring{\@strctr}}%
4986 \else
4987 \@modulo{\@strctr}{10}%
4988 \let\@@fc@numstr#2\relax
4989 \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
4990 \fi
4991 \fi
4992 \fi
4993 }

```

As above, but for ordinals

```

4994 \newcommand{\@@ordinalstringspanish}[2]{%
4995 \@strctr=#1\relax
4996 \ifnum#1>99999
4997 \PackageError{fmtcount}{Out of range}%

```

```

4998 {This macro only works for values less than 100000}%
4999 \else
5000 \ifnum#1<0
5001 \PackageError{fmtcount}{Negative numbers not permitted}%
5002 {This macro does not work for negative numbers, however
5003 you can try typing "minus" first, and then pass the modulus of
5004 this number}%
5005 \else
5006 \def#2{}%
5007 \ifnum \@strctr>999\relax
5008   \divide \@strctr by 1000\relax
5009   \ifnum \@strctr>1\relax
5010     \ifnum \@strctr>9\relax
5011       \@tmpstrctr=\@strctr
5012       \ifnum \@strctr<20
5013         \@modulo{\@tmpstrctr}{10}%
5014         \let\@@fc@ordstr#2\relax
5015         \edef#2{\@@fc@ordstr\@teenthstring{\@tmpstrctr}}%
5016       \else
5017         \divide \@tmpstrctr by 10\relax
5018         \let\@@fc@ordstr#2\relax
5019         \edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
5020         \@tmpstrctr=\@strctr
5021         \@modulo{\@tmpstrctr}{10}%
5022         \ifnum \@tmpstrctr>0\relax
5023           \let\@@fc@ordstr#2\relax
5024           \edef#2{\@@fc@ordstr\@unitthstring{\@tmpstrctr}}%
5025         \fi
5026       \fi
5027     \else
5028       \let\@@fc@ordstr#2\relax
5029       \edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
5030     \fi
5031   \fi
5032   \let\@@fc@ordstr#2\relax
5033   \edef#2{\@@fc@ordstr\@thousandth}%
5034 \fi
5035 \@strctr=#1\relax
5036 \@modulo{\@strctr}{1000}%
5037 \ifnum \@strctr>99\relax
5038   \@tmpstrctr=\@strctr
5039   \divide \@tmpstrctr by 100\relax
5040   \ifnum#1>1000\relax
5041     \let\@@fc@ordstr#2\relax
5042     \edef#2{\@@fc@ordstr\ }%
5043   \fi
5044   \let\@@fc@ordstr#2\relax
5045   \edef#2{\@@fc@ordstr\@hundredthstring{\@tmpstrctr}}%
5046 \fi

```



```

5047 \@modulo{\@strctr}{100}%
5048 \ifnum#1>99\relax
5049   \ifnum\@strctr>0\relax
5050     \let\@@fc@ordstr#2\relax
5051     \edef#2{\@@fc@ordstr\ }%
5052   \fi
5053 \fi
5054 \ifnum\@strctr>19\relax
5055   \@tmpstrctr=\@strctr
5056   \divide\@tmpstrctr by 10\relax
5057   \let\@@fc@ordstr#2\relax
5058   \edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
5059   \@tmpstrctr=\@strctr
5060   \@modulo{\@tmpstrctr}{10}%
5061   \ifnum\@tmpstrctr>0\relax
5062     \let\@@fc@ordstr#2\relax
5063     \edef#2{\@@fc@ordstr\ \@unitthstring{\@tmpstrctr}}%
5064   \fi
5065 \else
5066   \ifnum\@strctr>9\relax
5067     \@modulo{\@strctr}{10}%
5068     \let\@@fc@ordstr#2\relax
5069     \edef#2{\@@fc@ordstr\@teenthstring{\@strctr}}%
5070   \else
5071     \ifnum\@strctr=0\relax
5072       \ifnum#1=0\relax
5073         \let\@@fc@ordstr#2\relax
5074         \edef#2{\@@fc@ordstr\@unitstring{0}}%
5075       \fi
5076     \else
5077       \let\@@fc@ordstr#2\relax
5078       \edef#2{\@@fc@ordstr\@unitthstring{\@strctr}}%
5079     \fi
5080   \fi
5081 \fi
5082 \fi
5083 \fi
5084 }

```

9.4.14 fc-UKenglish.def

English definitions

```
5085 \ProvidesFCLanguage{UKenglish}[2012/06/18]
```

Loaded fc-english.def if not already loaded

```
5086 \FCloadlang{english}
```

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

```

5087 \let\@ordinalMUKenglish\@ordinalMenglish
5088 \let\@ordinalFUKenglish\@ordinalMenglish

```

```

5089 \let\@ordinalNUKenglish\@ordinalMenglish
5090 \let\@numberstringMUKenglish\@numberstringMenglish
5091 \let\@numberstringFUKenglish\@numberstringMenglish
5092 \let\@numberstringNUKenglish\@numberstringMenglish
5093 \let\@NumberstringMUKenglish\@NumberstringMenglish
5094 \let\@NumberstringFUKenglish\@NumberstringMenglish
5095 \let\@NumberstringNUKenglish\@NumberstringMenglish
5096 \let\@ordinalstringMUKenglish\@ordinalstringMenglish
5097 \let\@ordinalstringFUKenglish\@ordinalstringMenglish
5098 \let\@ordinalstringNUKenglish\@ordinalstringMenglish
5099 \let\@OrdinalstringMUKenglish\@OrdinalstringMenglish
5100 \let\@OrdinalstringFUKenglish\@OrdinalstringMenglish
5101 \let\@OrdinalstringNUKenglish\@OrdinalstringMenglish

```

9.4.15 fc-USenglish.def

US English definitions

```
5102 \ProvidesFCLanguage{USenglish}[2012/06/18]
```

Loaded fc-english.def if not already loaded

```
5103 \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.)

```

5104 \let\@ordinalMUSenglish\@ordinalMenglish
5105 \let\@ordinalFUSenglish\@ordinalMenglish
5106 \let\@ordinalNUSenglish\@ordinalMenglish
5107 \let\@numberstringMUSenglish\@numberstringMenglish
5108 \let\@numberstringFUSenglish\@numberstringMenglish
5109 \let\@numberstringNUSenglish\@numberstringMenglish
5110 \let\@NumberstringMUSenglish\@NumberstringMenglish
5111 \let\@NumberstringFUSenglish\@NumberstringMenglish
5112 \let\@NumberstringNUSenglish\@NumberstringMenglish
5113 \let\@ordinalstringMUSenglish\@ordinalstringMenglish
5114 \let\@ordinalstringFUSenglish\@ordinalstringMenglish
5115 \let\@ordinalstringNUSenglish\@ordinalstringMenglish
5116 \let\@OrdinalstringMUSenglish\@OrdinalstringMenglish
5117 \let\@OrdinalstringFUSenglish\@OrdinalstringMenglish
5118 \let\@OrdinalstringNUSenglish\@OrdinalstringMenglish

```