

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

Nicola L.C. Talbot

Vincent Belaïche

www.dickimaw-books.com

2024-08-31 (version 3.08)

Contents

1	Introduction	2
2	Available Commands	2
3	Package Options	8
4	Multilingual Support	8
4.1	Options for setting ordinal ending position raise/level	9
4.2	Options for French	10
4.3	Prefixes	14
5	Configuration File <code>fmtcount.cfg</code>	14
6	LaTeX2HTML style	15
7	Miscellaneous	15
7.1	Handling of spaces with tailing optional argument	15
7.2	Macro naming conventions	16
8	Acknowledgements	16
9	Troubleshooting	16
10	The Code	16
10.1	Language definition files	16
10.1.1	<code>fc-american.def</code>	16
10.1.2	<code>fc-brazilian.def</code>	17
10.1.3	<code>fc-british.def</code>	19
10.1.4	<code>fc-dutch.def</code>	19
10.1.5	<code>fc-english.def</code>	27

10.1.6 fc-francais.def	37
10.1.7 fc-french.def	37
10.1.8 fc-frenchb.def	69
10.1.9 fc-german.def	70
10.1.10fc-germanb.def	80
10.1.11fc-italian	80
10.1.12fc-ngerman.def	82
10.1.13fc-ngermanb.def	82
10.1.14fc-portuges.def	83
10.1.15fc-portuguese.def	98
10.1.16fc-spanish.def	98
10.1.17fc-UKenglish.def	115
10.1.18fc-USenglish.def	116
10.2 fcnumparser.sty	116
10.3 fcprefix.sty	127
10.4 fmtcount.sty	137
10.4.1 Multilanguage Definitions	162

1 Introduction

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

2 Available Commands

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

`\ordinal`

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

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

`\fmtord`

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

is used to format the `st`, `nd`, `rd`, `th` bit. By default the ordinal is formatted as a superscript, if the package option `level` is used, it is level with the text. For example, if the current section is 2, then `\ordinal{section}` will produce the output: 2nd. 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 `fntcount` package with the memoir class you should use

`\FCordinal`

```
\FCordinal
```

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

2. When the [`gender`] optional argument is omitted, no ignoring of spaces following the final argument occurs. So both `\ordinal{section}_!` and `\ordinal{section}[m]_!` will produce: 2nd_, where _ denotes a space. See § 7.1.

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{2}` will produce: 2nd.

`\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: Two.

`\NUMBERstring`

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

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

`\numberstringnum`

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

`\Numberstringnum`

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

`\NUMBERstringnum`

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

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

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

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

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

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

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

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

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

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

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

`RDINALstringnum` `\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{2}` will produce: second.

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>]
reordinalstring	\storeordinalstring{<label>}{<counter>}[<gender>]
reOrdinalstring	\storeOrdinalstring{<label>}{<counter>}[<gender>]
reORDINALstring	\storeORDINALstring{<label>}{<counter>}[<gender>]
renumberstring	\storenumberstring{<label>}{<counter>}[<gender>]
reNumberstring	\storeNumberstring{<label>}{<counter>}[<gender>]
reNUMBERstring	\storeNUMBERstring{<label>}{<counter>}[<gender>]
storeordinalnum	\storeordinalnum{<label>}{<number>}[<gender>]
reordinalstringnum	\storeordinalstring{<label>}{<number>}[<gender>]
reOrdinalstringnum	\storeOrdinalstringnum{<label>}{<number>}[<gender>]
reORDINALstringnum	\storeORDINALstringnum{<label>}{<number>}[<gender>]
renumberstringnum	\storenumberstring{<label>}{<number>}[<gender>]
reNumberstringnum	\storeNumberstring{<label>}{<number>}[<gender>]
reNUMBERstringnum	\storeNUMBERstring{<label>}{<number>}[<gender>]

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

This will print the value of *<counter>* as a binary number. E.g. `\binary{section}` will produce: 10. 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: 00000010. The default value for *<n>* is 17.

`\binarynum` `\binarynum{<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.

`\Hexadecimal` The macro `\Hexadecimal` is a deprecated alias of `\HEXADecimal`. Its name was confusing so it was changed. See [7.2](#).

`\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.

`\Hexadecimalnum`

The macro `\Hexadecimalnum` is a deprecated alias of `\HEXADecimalnum`. Its name was confusing so it was changed. See [7.2](#).

`\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: 00000002 still assuming current section is section 2.

`\decimalnum`

`\decimalnum{<n>}`

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

`\aaalph`

`\aaalph{<counter>}`

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

`\AAAAlph`

`\AAAAlph{<counter>}`

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

`\aaalphnum`

`\aaalphnum{<n>}`

`\AAAAlphnum`

`\AAAAlphnum{<n>}`

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

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

`\abalph`

`\abalph{<counter>}`

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

`\ABAlph` `\ABAlph{<counter>}`

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

`\abalphnum` `\abalphnum{<n>}`

`\ABAlphnum` `\ABAlphnum{<n>}`

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

3 Package Options

The following options can be passed to this package:

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

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

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

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

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

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

4 Multilingual Support

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

Actually, `fmtcount` has two modes:

- a multilingual mode, in which the commands `\numberstring`, `\ordinalstring`, `\ordinal`, and their variants will be formatted in the currently selected language, as per the `\language` macro set by `babel`, `polyglossia` or `suchlikes`, and

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

³Thanks to Edoardo Pasca for supplying the information.

- a default mode for backward compatibility in which these commands are formatted in English irrespective of `\languagename`, and to which `fmtcount` falls back when it cannot detect packages such as `babel` or `polyglossia` are loaded.

For multilingual mode, `fmtcount` needs to load correctly the language definition for document dialects. To do this use

`\FCloadlang`

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

in the preamble — this will both switch on multilingual mode, and load the `<dialect>` definition. The `<dialect>` should match the options passed to `babel` or `polyglossia`. `fmtcount` currently supports the following `<dialect>`'s: `english`, `UKenglish`, `brazilian`, `british`, `USenglish`, `american`, `spanish`, `portuges`, `portuguese`, `french`, `frenchb`, `francais`, `german`, `germanb`, `ngerman`, `ngermanb`, `italian`, and `dutch`.

If you don't use `\FCloadlang`, `fmtcount` will attempt to detect the required dialects and call `\FCloadlang` for you, but this isn't guaranteed to work. Notably, when `\FCloadlang` is not used and `fmtcount` has switched on multilingual mode, but without detecting the needed dialects in the preamble, and `fmtcount` has to format a number for a dialect for which definition has not been loaded (via `\FCloadlang` above), then if `fmtcount` detects a definition file for this dialect it will attempt to load it, and cause an error otherwise. This loading in body has not been tested extensively, and may cause problems such as spurious spaces insertion before the first formatted number, so it's best to use `\FCloadlang` explicitly in the preamble.

If the French language is selected, the `french` option let you configure the dialect and other aspects. The `abbr` also has some influence with French. Please refer to § 4.2.

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

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

4.1 Options for setting ordinal ending position raise/level

`countsetoptions`

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

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

⁴see § 3

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

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

4.2 Options for French

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

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

countsetoptions

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

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

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

dialect

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

french

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

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

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

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

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

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

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

abbr

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

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

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

L'option générale `abbr` permet de changer l'effet de `\ordinal`. Selon $\langle boolean \rangle$ on a :
`true` pour produire des ordinaux de la forme 2^e (par défaut), ou
`false` pour produire des ordinaux de la forme 2^{ème}

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$\langle x \rangle$ dans " $\langle x \rangle$ 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 $\langle scale \rangle$ à :

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

Par défaut, l'option vaut `recursive`.

n-illiard upto

```
\fmtcountsetoptions{french={n-illiard upto=\langle n-illiard upto \rangle}}
```

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

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

mil plural mark

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

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

4.3 Prefixes

latinnumeralstring

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

latinnumeralstringnum

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

5 Configuration File `fmtcount.cfg`

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

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

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

and if `fmtcount.cfg` has the line:

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

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

6 LaTeX2HTML style

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

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

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

This, I agree, is an unpleasant cludge.

7 Miscellaneous

7.1 Handling of spaces with tailing optional argument

Quite some of the commands in `fmtcount` have a tailing optional argument, notably a [*gender*] argument, which is due to historical reasons, and is a little unfortunate.

When the tailing optional argument is omitted, then any subsequent space will:

- not be gobbled if the command make some typset output, like `\ordinal` or `\numbestring`, and
- be gobbled if the command stores a number into a label like `\storeordinalnum` or `\storenumberstring`, or make some other border effect like `\padzeroes` without any typeset output.

So (where we use visible spaces “`□`” to demonstrate the point):

- “`x\ordinalnum{2}□x`” will be typeset to “`x2nd□x`”, while
- “`x\storeodinalnum{mylabel}{2}□x`” will be typeset to “`xx`”.

The reason for this design choice is that the commands like `\ordinal` or `\numbestring` are usually inserted in the flow of text, and one usually does not want subsequent spaces gobbled, while the commands like `\storeordinalnum` or `\storenumberstring` usually stands on their own line, and one usually does not want the tailing end-of-line to produce an extra-space.

7.2 Macro naming conventions

Macros that refer to upper-casing have upper case only in the main part of their name. That is to say the words “store”, “string” or “num” are not upper-cased for instance in `\storeORDINALstringnum`, `\storeOrdinalstringnum` or in `\NUMBERstringnum`.

Furthermore, when upper-casing all the number letters is considered, the main part of the name is:

- all in upper-case when it consist of a single word that is not composed of a prefix+radix, for instance “ORDINAL” or “NUMBER”, and
- with the prefix all in upper-case, and only the first letter of the radix in upper-case for words that consist of a prefix+radix, for instance “HEXADecimal” or “AAAlph” because they can be considered as a prefix+radix construct “hexa+decimal” or “aa+alph”.

Observance of this rule is the reason why macros `\Hexadecimal` and `\Hexadecimalnum` were respectively renamed as `\HEXADecimal` and `\HEXADecimalnum` from v3.06.

8 Acknowledgements

I would like to thank all the people who have provided translations and made bug reports.

9 Troubleshooting

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

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

Local Variables: coding: utf-8 compile-command: "make -C ../dist fmtcount.pdf" End:

10 The Code

10.1 Language definition files

10.1.1 fc-american.def

American English definitions

```
1 \ProvidesFCLanguage{american}[2016/01/12]%
```

Loaded fc-USenglish.def if not already loaded

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

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

```
3 \global\let\@ordinalMamerican\@ordinalMUSenglish
```

```
4 \global\let\@ordinalFamerican\@ordinalMUSenglish
```

```
5 \global\let\@ordinalNamerican\@ordinalMUSenglish
```

```
6 \global\let\@numberstringMamerican\@numberstringMUSenglish
```



```

7 \global\let\@numberstringFamerican\@numberstringMUSenglish
8 \global\let\@numberstringNamerican\@numberstringMUSenglish
9 \global\let\@NumberstringMamerican\@NumberstringMUSenglish
10 \global\let\@NumberstringFamerican\@NumberstringMUSenglish
11 \global\let\@NumberstringNamerican\@NumberstringMUSenglish
12 \global\let\@ordinalstringMamerican\@ordinalstringMUSenglish
13 \global\let\@ordinalstringFamerican\@ordinalstringMUSenglish
14 \global\let\@ordinalstringNamerican\@ordinalstringMUSenglish
15 \global\let\@OrdinalstringMamerican\@OrdinalstringMUSenglish
16 \global\let\@OrdinalstringFamerican\@OrdinalstringMUSenglish
17 \global\let\@OrdinalstringNamerican\@OrdinalstringMUSenglish

```

10.1.2 fc-brazilian.def

Brazilian definitions.

```
18 \ProvidesFCLanguage{brazilian}[2017/12/26]%
```

Load fc-portuges.def if not already loaded.

```
19 \FCloadlang{portuges}%
```

Set |brazilian| to be equivalent to |portuges| for all the numeral ordinals, and string ordinals.

```

20 \global\let\@ordinalMbrazilian=\@ordinalMportuges
21 \global\let\@ordinalFbrazilian=\@ordinalFportuges
22 \global\let\@ordinalNbrazilian=\@ordinalNportuges
23 \global\let\@ordinalstringFbrazilian\@ordinalstringFportuges
24 \global\let\@ordinalstringMbrazilian\@ordinalstringMportuges
25 \global\let\@ordinalstringNbrazilian\@ordinalstringMportuges
26 \global\let\@OrdinalstringMbrazilian\@OrdinalstringMportuges
27 \global\let\@OrdinalstringFbrazilian\@OrdinalstringFportuges
28 \global\let\@OrdinalstringNbrazilian\@OrdinalstringMportuges

```

Convert a number to a textual representation. To make it easier, split it up into units, tens, teens and hundreds. Units, tens, and hundreds are the same as for |portuges| and are not redefined, only the teens are Brazilian specific.

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

```

29 \newcommand*{\@teenstringbrazilian[1]}{
30   \ifcase#1\relax
31     dez%
32     \or onze%
33     \or doze%
34     \or treze%
35     \or quatorze%
36     \or quinze%
37     \or dezesseis%
38     \or dezessete%
39     \or dezoito%
40     \or dezenove%
41   \fi
42 }%
43 \global\let\@teenstringbrazilian\@teenstringbrazilian

```

Teens (with initial letter in upper case):

```
44 \newcommand*\@@Teenstringbrazilian[1]{%
45   \ifcase#1\relax
46     Dez%
47     \or Onze%
48     \or Doze%
49     \or Treze%
50     \or Quatorze%
51     \or Quinze%
52     \or Dezesesseis%
53     \or Dezesesete%
54     \or Dezoito%
55     \or Dezenove%
56   \fi
57 }%
58 \global\let\@@Teenstringbrazilian\@@Teenstringbrazilian
```

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.)

```
59 \newcommand*\@numberstringMbrazilian}[2]{%
60   \let\@unitstring=\@unitstringportuges
61   \let\@teenstring=\@teenstringbrazilian
62   \let\@tenstring=\@tenstringportuges
63   \let\@hundredstring=\@hundredstringportuges
64   \def\@hundred{cem}\def\@thousand{mil}%
65   \def\@andname{e}%
66   \@numberstringportuges{#1}{#2}%
67 }%
68 \global\let\@numberstringMbrazilian\@numberstringMbrazilian
```

As above, but feminine form:

```
69 \newcommand*\@numberstringFbrazilian}[2]{%
70   \let\@unitstring=\@unitstringFportuges
71   \let\@teenstring=\@teenstringbrazilian
72   \let\@tenstring=\@tenstringportuges
73   \let\@hundredstring=\@hundredstringFportuges
74   \def\@hundred{cem}\def\@thousand{mil}%
75   \def\@andname{e}%
76   \@numberstringportuges{#1}{#2}%
77 }%
78 \global\let\@numberstringFbrazilian\@numberstringFbrazilian
```

Make neuter same as masculine:

```
79 \global\let\@numberstringNbrazilian\@numberstringMbrazilian
```

As above, but initial letters in upper case:

```
80 \newcommand*\@NumberstringMbrazilian}[2]{%
81   \let\@unitstring=\@unitstringportuges
82   \let\@teenstring=\@Teenstringbrazilian
83   \let\@tenstring=\@Tenstringportuges
```

```

84 \let\@hundredstring=\@hundredstringportuges
85 \def\@hundred{Cem}\def\@thousand{Mil}%
86 \def\@andname{e}%
87 \@numberstringportuges{#1}{#2}%
88 }%
89 \global\let\@NumberstringMbrazilian\@NumberstringMbrazilian

```

As above, but feminine form:

```

90 \newcommand*\@NumberstringFbrazilian}[2]{%
91 \let\@unitstring=\@UnitstringFportuges
92 \let\@teenstring=\@Teenstringbrazilian
93 \let\@tenstring=\@Tenstringportuges
94 \let\@hundredstring=\@HundredstringFportuges
95 \def\@hundred{Cem}\def\@thousand{Mil}%
96 \def\@andname{e}%
97 \@numberstringportuges{#1}{#2}%
98 }%
99 \global\let\@NumberstringFbrazilian\@NumberstringFbrazilian

```

Make neuter same as masculine:

```

100 \global\let\@NumberstringNbrazilian\@NumberstringMbrazilian

```

10.1.3 fc-british.def

British definitions

```

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

```

Load fc-english.def, if not already loaded

```

102 \FCloadlang{english}%

```

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

```

103 \global\let\@ordinalMbritish\@ordinalMenglish
104 \global\let\@ordinalFbritish\@ordinalMenglish
105 \global\let\@ordinalNbritish\@ordinalMenglish
106 \global\let\@numberstringMbritish\@numberstringMenglish
107 \global\let\@numberstringFbritish\@numberstringMenglish
108 \global\let\@numberstringNbritish\@numberstringMenglish
109 \global\let\@NumberstringMbritish\@NumberstringMenglish
110 \global\let\@NumberstringFbritish\@NumberstringMenglish
111 \global\let\@NumberstringNbritish\@NumberstringMenglish
112 \global\let\@ordinalstringMbritish\@ordinalstringMenglish
113 \global\let\@ordinalstringFbritish\@ordinalstringMenglish
114 \global\let\@ordinalstringNbritish\@ordinalstringMenglish
115 \global\let\@OrdinalstringMbritish\@OrdinalstringMenglish
116 \global\let\@OrdinalstringFbritish\@OrdinalstringMenglish
117 \global\let\@OrdinalstringNbritish\@OrdinalstringMenglish

```

10.1.4 fc-dutch.def

Dutch definitions, initially added by Erik Nijenhuis.

```

118 \ProvidesFCLanguage{dutch}[2024/01/27]%

```

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.

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

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

```
121 \global\let\@ordinalFdutch\@ordinalMdutch
122 \global\let\@ordinalNdutch\@ordinalMdutch
```

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.

```
123 \newcommand*\@@unitstringdutch[1]{%
124   \ifcase#1%
125   nul%
126   \or een% één and \'e\'en not working atm
127   \or twee%
128   \or drie%
129   \or vier%
130   \or vijf%
131   \or zes%
132   \or zeven%
133   \or acht%
134   \or negen%
135   \fi
136 }%
137 \global\let\@@unitstringdutch\@@unitstringdutch
```

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

```
138 \global\let\@@unitstringdutch\@@unitstringdutch
139 \newcommand*\@@tenstringdutch[1]{%
140   \ifcase#1%
141   \or tien%
142   \or twintig%
143   \or dertig%
144   \or veertig%
145   \or vijftig%
146   \or zestig%
147   \or zeventig%
148   \or tachtig%
149   \or negentig%
150   \or honderd%
151   \fi
152 }%
153 \global\let\@@tenstringdutch\@@tenstringdutch
```

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

```
154 \newcommand*\@@teenstringdutch[1]{%
155   \ifcase#1%
```

```

156     tien%
157     \or elf%
158     \or twaalf%
159     \or dertien%
160     \or veertien%
161     \or vijftien%
162     \or zestien%
163     \or zeventien%
164     \or achttien%
165     \or negentien%
166     \fi
167 }%
168 \global\let\@@teenstringdutch\@@teenstringdutch

Hunderd and thousand:
169 \providecommand*\honderd{\honderd}%
170 \providecommand*\duizend{\duizend}%
171 \global\let\honderd\honderd
172 \global\let\duizend\duizend

The numberstring implementation:
173 \newcommand*\@@numberstringdutch[2]{%
174     \ifnum#1>99999\relax
175     \PackageError{fmtcount}{Out of range}%
176     {This macro only works for values less than 100000}%
177     \else
178     \ifnum#1<0\relax
179     \PackageError{fmtcount}{Negative numbers not permitted}%
180     {This macro does not work for negative numbers, however
181     you can try typing "minus" first, and then pass the modulus of
182     this number}%
183     \fi
184     \fi
185     \def#2{}%
186     \@strctr=#1\relax \divide\@strctr by 1000\relax
187     \ifnum\@strctr>1\relax
188     \@@numberunderhundreddutch{\@strctr}{#2}%
189     \appto#2{\duizend}%
190     \else
191     \ifnum\@strctr=1\relax
192     \eappto#2{\duizend}%
193     \fi
194     \fi
195     \@strctr=#1\relax
196     \@FCmodulo{\@strctr}{1000}%
197     \divide\@strctr by 100\relax
198     \ifnum\@strctr>1\relax
199     \eappto#2{\@unitstring{\@strctr}honderd}%
200     \else
201     \ifnum\@strctr=1\relax

```

```

202 \ifnum#1>1000\relax
203 \appto#2{honderd}%
204 \else
205 \eappto#2{\honderd}%
206 \fi
207 \fi
208 \fi
209 \@strctr=#1\relax
210 \@FCmodulo{\@strctr}{100}%
211 \ifnum#1=0\relax
212 \def#2{null}%
213 \else
214 \ifnum\@strctr=1\relax
215 \appto#2{een}% één and \’e\’en not working atm
216 \else
217 \@@numberunderhundreddutch{\@strctr}{#2}%
218 \fi
219 \fi
220 }%
221 \global\let\@@numberstringdutch\@@numberstringdutch

```

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

```

222 \newcommand*\@numberstringMdutch}[2]{%
223 \let\@unitstring=\@unitstringdutch%
224 \let\@teenstring=\@teenstringdutch%
225 \let\@tenstring=\@tenstringdutch%
226 \def\@hundred{honderd}\def\@thousand{duizend}%
227 \@@numberstringdutch{#1}{#2}%
228 }%
229 \global\let\@numberstringMdutch\@numberstringMdutch

```

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

```

230 \global\let\@numberstringFdutch=\@numberstringMdutch
231 \global\let\@numberstringNdutch=\@numberstringMdutch

```

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

```

232 \newcommand*\@NumberstringMdutch}[2]{%
233 \@numberstringMdutch{#1}{\@@num@str}%
234 \edef#2{\noexpand\MakeUppercase\expandonce\@@num@str}%
235 }%
236 \global\let\@NumberstringMdutch\@NumberstringMdutch

```

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

```

237 \global\let\@NumberstringFdutch=\@NumberstringMdutch
238 \global\let\@NumberstringNdutch=\@NumberstringMdutch

```

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

```

239 \newcommand*\@unitthstringdutch[1]{%
240 \ifcase#1%

```

```

241     nulde%
242     \or eerste% éérste and \'e\'erste not working atm
243     \or tweede%
244     \or derde%
245     \or vierde%
246     \or vijfde%
247     \or zesde%
248     \or zevende%
249     \or achtste%
250     \or negende%
251     \fi
252 }%
253 \global\let\@@unitthstringdutch\@@unitthstringdutch

```

Next the tens:

```

254 \newcommand*\@@tenthstringdutch[1]{%
255     \ifcase#1%
256     \or tiende%
257     \or twintigste%
258     \or dertigste%
259     \or veertigste%
260     \or vijftigste%
261     \or zestigste%
262     \or zeventigste%
263     \or tachtigste%
264     \or negentigste%
265     \fi
266 }%
267 \global\let\@@tenthstringdutch\@@tenthstringdutch

```

The teens:

```

268 \newcommand*\@@teenthstringdutch[1]{%
269     \ifcase#1%
270     tiende%
271     \or elfde%
272     \or twaalfde%
273     \or dertiende%
274     \or veertiende%
275     \or vijftiende%
276     \or zestiende%
277     \or zeventiende%
278     \or achttiende%
279     \or negentiende%
280     \fi
281 }%
282 \global\let\@@teenthstringdutch\@@teenthstringdutch

```

The ordinalstring implementation:

```

283 \newcommand*\@@ordinalstringdutch[2]{%
284     \@orgargctr=#1\relax
285     \ifnum\@orgargctr>99999\relax

```

```

286 \PackageError{fmtcount}{Out of range}%
287 {This macro only works for values less than 100000}%
288 \else
289 \ifnum \@orgargctr<0\relax
290 \PackageError{fmtcount}{Negative numbers not permitted}%
291 {This macro does not work for negative numbers, however
292 you can try typing "minus" first, and then pass the modulus of
293 this number}%
294 \fi
295 \fi
296 \def#2{}%
297 \@strctr=\@orgargctr\divide\@strctr by 1000\relax
298 \ifnum \@strctr>1\relax
299 \@numberunderhundreddutch{\@strctr}{#2}%
300 \@tmpstrctr=\@orgargctr \@FCmodulo{\@tmpstrctr}{1000}%
301 \ifnum \@tmpstrctr=0\relax
302 \eappto#2{\@thousandth}%
303 \else
304 \appto#2{duizend}%
305 \fi
306 \else
307 \ifnum \@strctr=1\relax
308 \ifnum \@orgargctr=1000\relax
309 \eappto#2{\@thousandth}%
310 \else
311 \eappto#2{\duizend}%
312 \fi
313 \fi
314 \fi
315 \@strctr=\@orgargctr%
316 \@FCmodulo{\@strctr}{1000}%
317 \divide\@strctr by 100\relax
318 \ifnum \@strctr>1\relax
319 \@tmpstrctr=\@orgargctr \@FCmodulo{\@tmpstrctr}{100}%
320 \ifnum \@tmpstrctr=0\relax
321 \ifnum \@strctr=1\relax
322 \eappto#2{\@hundredth}%
323 \else
324 \eappto#2{\@unitstring{\@strctr}\@hundredth}%
325 \fi
326 \else
327 \eappto#2{\@unitstring{\@strctr}honderd}%
328 \fi
329 \else
330 \ifnum \@strctr=1\relax
331 \@tmpstrctr=\@orgargctr \@FCmodulo{\@tmpstrctr}{100}%
332 \ifnum \@tmpstrctr=0\relax
333 \eappto#2{\@hundredth}%
334 \else

```



```

335 \ifnum\@orgargctr>1000\relax
336 \appto#2{honderd}%
337 \else
338 \eappto#2{\honderd}%
339 \fi
340 \fi
341 \fi
342 \fi
343 \@strctr=\@orgargctr%
344 \@FCmodulo{\@strctr}{100}%
345 \ifthenelse{\@strctr=0 \and \@orgargctr>0 }{-}{%
346 \@@numberunderhundredthdutch{\@strctr}{#2}%
347 }%
348 }%
349 \global\let\@@ordinalstringdutch\@@ordinalstringdutch

```

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

```

350 \newcommand*{\@ordinalstringMdutch}[2]{%
351 \let\@unitthstring=\@unitthstringdutch%
352 \let\@teenthstring=\@teenthstringdutch%
353 \let\@tenthstring=\@tenthstringdutch%
354 \let\@unitstring=\@unitstringdutch%
355 \let\@teenstring=\@teenstringdutch%
356 \let\@tenstring=\@tenstringdutch%
357 \def\@thousandth{duizendste}%
358 \def\@hundredth{honderdste}%
359 \@ordinalstringdutch{#1}{#2}%
360 }%
361 \global\let\@ordinalstringMdutch\@ordinalstringMdutch

```

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

```

362 \global\let\@ordinalstringFdutch=\@ordinalstringMdutch
363 \global\let\@ordinalstringNdutch=\@ordinalstringMdutch

```

First letter of each word in upper case:

```

364 \newcommand*{\@OrdinalstringMdutch}[2]{%
365 \@ordinalstringMdutch{#1}{\@num@str}%
366 \def\@hundred{Honderd}\def\@thousand{Duizend}%
367 \def\@hundredth{Honderdste}\def\@thousandth{Duizendste}%
368 \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
369 }%
370 \global\let\@OrdinalstringMdutch\@OrdinalstringMdutch

```

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

```

371 \global\let\@OrdinalstringFdutch=\@OrdinalstringMdutch
372 \global\let\@OrdinalstringNdutch=\@OrdinalstringMdutch

```

For numbers under hunderd:

```

373 \newcommand*{\@@numberunderhundreddutch}[2]{%
374 \ifnum#1<10\relax

```

```

375 \ifnum#1>0\relax
376 \eappto#2{\@unitstring{#1}}%
377 \fi
378 \else
379 \@tmpstrctr=#1\relax
380 \@FCmodulo{\@tmpstrctr}{10}%
381 \ifnum#1<20\relax
382 \eappto#2{\@teenstring{\@tmpstrctr}}%
383 \else
384 \ifnum\@tmpstrctr=0\relax
385 \else

```

For digits ending with an ‘e’, a trema gets added for \@andname. Take for example drieën-twintig or tweeënveertig.

```

386 \ifnum\@tmpstrctr=2\relax\def\@andname{ën}%
387 \else\ifnum\@tmpstrctr=3\relax\def\@andname{ën}%
388 \else\def\@andname{en}%
389 \fi\fi%
390 \eappto#2{\@unitstring{\@tmpstrctr}\@andname}%
391 \fi
392 \@tmpstrctr=#1\relax
393 \divide\@tmpstrctr by 10\relax
394 \eappto#2{\@tenstring{\@tmpstrctr}}%
395 \fi
396 \fi
397 }%
398 \global\let\@@numberunderhundreddutch\@@numberunderhundreddutch
399 \newcommand*{\@@numberunderhundreddthdutch}[2]{%
400 \ifnum#1<10\relax
401 \eappto#2{\@unitthstring{#1}}%
402 \else
403 \@tmpstrctr=#1\relax
404 \@FCmodulo{\@tmpstrctr}{10}%
405 \ifnum#1<20\relax
406 \eappto#2{\@teenthstring{\@tmpstrctr}}%
407 \else
408 \ifnum\@tmpstrctr=0\relax
409 \else

```

Again, for digits ending with an ‘e’, a trema gets added for \@andname (drieëntwintig or tweeënveertig).

```

410 \ifnum\@tmpstrctr=2\relax\def\@andname{ën}%
411 \else\ifnum\@tmpstrctr=3\relax\def\@andname{ën}%
412 \else\def\@andname{en}%
413 \fi\fi%
414 \eappto#2{\@unitstring{\@tmpstrctr}\@andname}%
415 \fi
416 \@tmpstrctr=#1\relax
417 \divide\@tmpstrctr by 10\relax
418 \eappto#2{\@tenthstring{\@tmpstrctr}}%

```

```

419 \fi
420 \fi
421 }%
422 \global\let\@@numberunderhundredthdutch\@@numberunderhundredthdutch

```

10.1.5 fc-english.def

English definitions

```
423 \ProvidesFCLanguage{english}[2016/01/12]%
```

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.

```

424 \newcommand*\@ordinalMenglish[2]{%
425 \def\@fc@ord{}%
426 \@orgargctr=#1\relax
427 \@ordinalctr=#1%
428 \@FCmodulo{\@ordinalctr}{100}%
429 \ifnum\@ordinalctr=11\relax
430 \def\@fc@ord{th}%
431 \else
432 \ifnum\@ordinalctr=12\relax
433 \def\@fc@ord{th}%
434 \else
435 \ifnum\@ordinalctr=13\relax
436 \def\@fc@ord{th}%
437 \else
438 \@FCmodulo{\@ordinalctr}{10}%
439 \ifcase\@ordinalctr
440 \def\@fc@ord{th}% case 0
441 \or \def\@fc@ord{st}% case 1
442 \or \def\@fc@ord{nd}% case 2
443 \or \def\@fc@ord{rd}% case 3
444 \else
445 \def\@fc@ord{th}% default case
446 \fi
447 \fi
448 \fi
449 \fi
450 \edef#2{\number#1\relax\noexpand\fmtord{\@fc@ord}}%
451 }%
452 \global\let\@ordinalMenglish\@ordinalMenglish

```

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

```

453 \global\let\@ordinalFenglish=\@ordinalMenglish
454 \global\let\@ordinalNenglish=\@ordinalMenglish

```

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

```

455 \newcommand*\@@unitstringenglish[1]{%
456   \ifcase#1\relax
457     zero%
458     \or one%
459     \or two%
460     \or three%
461     \or four%
462     \or five%
463     \or six%
464     \or seven%
465     \or eight%
466     \or nine%
467 \fi
468 }%
469 \global\let\@@unitstringenglish\@@unitstringenglish

```

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

```

470 \newcommand*\@@tenstringenglish[1]{%
471   \ifcase#1\relax
472     \or ten%
473     \or twenty%
474     \or thirty%
475     \or forty%
476     \or fifty%
477     \or sixty%
478     \or seventy%
479     \or eighty%
480     \or ninety%
481 \fi
482 }%
483 \global\let\@@tenstringenglish\@@tenstringenglish

```

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

```

484 \newcommand*\@@teenstringenglish[1]{%
485   \ifcase#1\relax
486     ten%
487     \or eleven%
488     \or twelve%
489     \or thirteen%
490     \or fourteen%
491     \or fifteen%
492     \or sixteen%
493     \or seventeen%
494     \or eighteen%
495     \or nineteen%
496 \fi
497 }%
498 \global\let\@@teenstringenglish\@@teenstringenglish

```

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

```

499 \newcommand*\@@Unitstringenglish[1]{%

```

```

500 \ifcase#1\relax
501   Zero%
502   \or One%
503   \or Two%
504   \or Three%
505   \or Four%
506   \or Five%
507   \or Six%
508   \or Seven%
509   \or Eight%
510   \or Nine%
511 \fi
512 }%
513 \global\let\@@Unitstringenglish\@@Unitstringenglish

```

The tens:

```

514 \newcommand*\@@Tenstringenglish[1]{%
515   \ifcase#1\relax
516   \or Ten%
517   \or Twenty%
518   \or Thirty%
519   \or Forty%
520   \or Fifty%
521   \or Sixty%
522   \or Seventy%
523   \or Eighty%
524   \or Ninety%
525 \fi
526 }%
527 \global\let\@@Tenstringenglish\@@Tenstringenglish

```

The teens:

```

528 \newcommand*\@@Teenstringenglish[1]{%
529   \ifcase#1\relax
530   Ten%
531   \or Eleven%
532   \or Twelve%
533   \or Thirteen%
534   \or Fourteen%
535   \or Fifteen%
536   \or Sixteen%
537   \or Seventeen%
538   \or Eighteen%
539   \or Nineteen%
540 \fi
541 }%
542 \global\let\@@Teenstringenglish\@@Teenstringenglish

```

This has changed in version 1.09, so that it now stores the result in the second argument, but doesn't display anything. Since it only affects internal macros, it shouldn't affect documents

created with older versions. (These internal macros are not meant for use in documents.)

```
543 \newcommand*{\@@numberstringenglish[2]}{%
544 \ifnum#1>99999
545 \PackageError{fmtcount}{Out of range}%
546 {This macro only works for values less than 100000}%
547 \else
548 \ifnum#1<0
549 \PackageError{fmtcount}{Negative numbers not permitted}%
550 {This macro does not work for negative numbers, however
551 you can try typing "minus" first, and then pass the modulus of
552 this number}%
553 \fi
554 \fi
555 \def#2{}%
556 \@strctr=#1\relax \divide\@strctr by 1000\relax
557 \ifnum\@strctr>9
558 \divide\@strctr by 10
559 \ifnum\@strctr>1\relax
560 \let\@@fc@numstr#2\relax
561 \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
562 \@strctr=#1 \divide\@strctr by 1000\relax
563 \@FCmodulo{\@strctr}{10}%
564 \ifnum\@strctr>0\relax
565 \let\@@fc@numstr#2\relax
566 \edef#2{\@@fc@numstr-\@unitstring{\@strctr}}%
567 \fi
568 \else
569 \@strctr=#1\relax
570 \divide\@strctr by 1000\relax
571 \@FCmodulo{\@strctr}{10}%
572 \let\@@fc@numstr#2\relax
573 \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
574 \fi
575 \let\@@fc@numstr#2\relax
576 \edef#2{\@@fc@numstr\ \@thousand}%
577 \else
578 \ifnum\@strctr>0\relax
579 \let\@@fc@numstr#2\relax
580 \edef#2{\@@fc@numstr\@unitstring{\@strctr}\ \@thousand}%
581 \fi
582 \fi
583 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
584 \divide\@strctr by 100
585 \ifnum\@strctr>0\relax
586 \ifnum#1>1000\relax
587 \let\@@fc@numstr#2\relax
588 \edef#2{\@@fc@numstr\ }%
589 \fi
590 \let\@@fc@numstr#2\relax
```

```

591 \edef#2{\@fc@numstr\@unitstring{\@strctr}\ \@hundred}%
592 \fi
593 \@strctr=#1\relax \@FCmodulo{\@strctr}{100}%
594 \ifnum#1>100\relax
595 \ifnum\@strctr>0\relax
596 \let\@fc@numstr#2\relax
597 \edef#2{\@fc@numstr\ \@andname\ }%
598 \fi
599 \fi
600 \ifnum\@strctr>19\relax
601 \divide\@strctr by 10\relax
602 \let\@fc@numstr#2\relax
603 \edef#2{\@fc@numstr\@tenstring{\@strctr}}%
604 \@strctr=#1\relax \@FCmodulo{\@strctr}{10}%
605 \ifnum\@strctr>0\relax
606 \let\@fc@numstr#2\relax
607 \edef#2{\@fc@numstr-\@unitstring{\@strctr}}%
608 \fi
609 \else
610 \ifnum\@strctr<10\relax
611 \ifnum\@strctr=0\relax
612 \ifnum#1<100\relax
613 \let\@fc@numstr#2\relax
614 \edef#2{\@fc@numstr\@unitstring{\@strctr}}%
615 \fi
616 \else
617 \let\@fc@numstr#2\relax
618 \edef#2{\@fc@numstr\@unitstring{\@strctr}}%
619 \fi
620 \else
621 \@FCmodulo{\@strctr}{10}%
622 \let\@fc@numstr#2\relax
623 \edef#2{\@fc@numstr\@teenstring{\@strctr}}%
624 \fi
625 \fi
626 }%
627 \global\let\@numberstringenglish\@numberstringenglish

```

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

```

628 \newcommand*{\@numberstringMenglish}[2]{%
629 \let\@unitstring=\@unitstringenglish
630 \let\@teenstring=\@teenstringenglish
631 \let\@tenstring=\@tenstringenglish
632 \def\@hundred{hundred}\def\@thousand{thousand}%
633 \def\@andname{and}%
634 \@numberstringenglish{#1}{#2}%
635 }%
636 \global\let\@numberstringMenglish\@numberstringMenglish

```

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

```
637 \global\let\@numberstringFenglish=\@numberstringMenglish
638 \global\let\@numberstringNenglish=\@numberstringMenglish
```

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

```
639 \newcommand*\@NumberstringMenglish[2]{%
640   \let\@unitstring=\@Unitstringenglish
641   \let\@teenstring=\@Teenstringenglish
642   \let\@tenstring=\@Tenstringenglish
643   \def\@hundred{Hundred}\def\@thousand{Thousand}%
644   \def\@andname{and}%
645   \@numberstringenglish{#1}{#2}%
646 }%
647 \global\let\@NumberstringMenglish\@NumberstringMenglish
```

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

```
648 \global\let\@NumberstringFenglish=\@NumberstringMenglish
649 \global\let\@NumberstringNenglish=\@NumberstringMenglish
```

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

```
650 \newcommand*\@unitthstringenglish[1]{%
651   \ifcase#1\relax
652     zeroth%
653   \or first%
654   \or second%
655   \or third%
656   \or fourth%
657   \or fifth%
658   \or sixth%
659   \or seventh%
660   \or eighth%
661   \or ninth%
662   \fi
663 }%
664 \global\let\@unitthstringenglish\@unitthstringenglish
```

Next the tens:

```
665 \newcommand*\@tenthstringenglish[1]{%
666   \ifcase#1\relax
667     \or tenth%
668     \or twentieth%
669     \or thirtieth%
670     \or fortieth%
671     \or fiftieth%
672     \or sixtieth%
673     \or seventieth%
674     \or eightieth%
675     \or ninetieth%
676   \fi
677 }%
```



```
678 \global\let\@@tenthstringenglish\@@tenthstringenglish
```

The teens:

```
679 \newcommand*\@@teenthstringenglish[1]{%
```

```
680 \ifcase#1\relax
```

```
681 tenth%
```

```
682 \or eleventh%
```

```
683 \or twelfth%
```

```
684 \or thirteenth%
```

```
685 \or fourteenth%
```

```
686 \or fifteenth%
```

```
687 \or sixteenth%
```

```
688 \or seventeenth%
```

```
689 \or eighteenth%
```

```
690 \or nineteenth%
```

```
691 \fi
```

```
692 }%
```

```
693 \global\let\@@teenthstringenglish\@@teenthstringenglish
```

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

```
694 \newcommand*\@@Unitthstringenglish[1]{%
```

```
695 \ifcase#1\relax
```

```
696 Zeroth%
```

```
697 \or First%
```

```
698 \or Second%
```

```
699 \or Third%
```

```
700 \or Fourth%
```

```
701 \or Fifth%
```

```
702 \or Sixth%
```

```
703 \or Seventh%
```

```
704 \or Eighth%
```

```
705 \or Ninth%
```

```
706 \fi
```

```
707 }%
```

```
708 \global\let\@@Unitthstringenglish\@@Unitthstringenglish
```

The tens:

```
709 \newcommand*\@@Tenthstringenglish[1]{%
```

```
710 \ifcase#1\relax
```

```
711 \or Tenth%
```

```
712 \or Twentieth%
```

```
713 \or Thirtieth%
```

```
714 \or Fortieth%
```

```
715 \or Fiftieth%
```

```
716 \or Sixtieth%
```

```
717 \or Seventieth%
```

```
718 \or Eightieth%
```

```
719 \or Ninetieth%
```

```
720 \fi
```

```
721 }%
```

```
722 \global\let\@@Tenthstringenglish\@@Tenthstringenglish
```

The teens:

```
723 \newcommand*{\@@Teenthstringenglish[1]}{%
724   \ifcase#1\relax
725     Tenth%
726     \or Eleventh%
727     \or Twelfth%
728     \or Thirteenth%
729     \or Fourteenth%
730     \or Fifteenth%
731     \or Sixteenth%
732     \or Seventeenth%
733     \or Eighteenth%
734     \or Nineteenth%
735   \fi
736 }%
737 \global\let\@@Teenthstringenglish\@@Teenthstringenglish
```

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

```
738 \newcommand*{\@@ordinalstringenglish[2]}{%
739 \@strctr=#1\relax
740 \ifnum#1>99999
741 \PackageError{fmtcount}{Out of range}%
742 {This macro only works for values less than 100000 (value given: \number\@strctr)}%
743 \else
744 \ifnum#1<0
745 \PackageError{fmtcount}{Negative numbers not permitted}%
746 {This macro does not work for negative numbers, however
747 you can try typing "minus" first, and then pass the modulus of
748 this number}%
749 \fi
750 \def#2{}%
751 \fi
752 \@strctr=#1\relax \divide\@strctr by 1000\relax
753 \ifnum\@strctr>9\relax
754   #1 is greater or equal to 10000
755   \divide\@strctr by 10
756   \ifnum\@strctr>1\relax
757     \let\@@fc@ordstr#2\relax
758     \edef#2{\@@fc@ordstr\@tenstring{\@strctr}}%
759     \@strctr=#1\relax
760     \divide\@strctr by 1000\relax
761     \@FCmodulo{\@strctr}{10}%
762     \ifnum\@strctr>0\relax
763       \let\@@fc@ordstr#2\relax
764       \edef#2{\@@fc@ordstr-\@unitstring{\@strctr}}%
765     \fi
766   \else
```

```

766 \@strctr=#1\relax \divide\@strctr by 1000\relax
767 \@FCmodulo{\@strctr}{10}%
768 \let\@@fc@ordstr#2\relax
769 \edef#2{\@@fc@ordstr\@teenstring{\@strctr}}%
770 \fi
771 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
772 \ifnum\@strctr=0\relax
773 \let\@@fc@ordstr#2\relax
774 \edef#2{\@@fc@ordstr\ \@thousandth}%
775 \else
776 \let\@@fc@ordstr#2\relax
777 \edef#2{\@@fc@ordstr\ \@thousand}%
778 \fi
779 \else
780 \ifnum\@strctr>0\relax
781 \let\@@fc@ordstr#2\relax
782 \edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
783 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
784 \let\@@fc@ordstr#2\relax
785 \ifnum\@strctr=0\relax
786 \edef#2{\@@fc@ordstr\ \@thousandth}%
787 \else
788 \edef#2{\@@fc@ordstr\ \@thousand}%
789 \fi
790 \fi
791 \fi
792 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
793 \divide\@strctr by 100
794 \ifnum\@strctr>0\relax
795 \ifnum#1>1000\relax
796 \let\@@fc@ordstr#2\relax
797 \edef#2{\@@fc@ordstr\ }%
798 \fi
799 \let\@@fc@ordstr#2\relax
800 \edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
801 \@strctr=#1\relax \@FCmodulo{\@strctr}{100}%
802 \let\@@fc@ordstr#2\relax
803 \ifnum\@strctr=0\relax
804 \edef#2{\@@fc@ordstr\ \@hundredth}%
805 \else
806 \edef#2{\@@fc@ordstr\ \@hundred}%
807 \fi
808 \fi
809 \@strctr=#1\relax \@FCmodulo{\@strctr}{100}%
810 \ifnum#1>100\relax
811 \ifnum\@strctr>0\relax
812 \let\@@fc@ordstr#2\relax
813 \edef#2{\@@fc@ordstr\ \@andname\ }%
814 \fi

```

```

815 \fi
816 \ifnum \@strctr>19\relax
817 \@tmpstrctr=\@strctr
818 \divide \@strctr by 10\relax
819 \@FCmodulo{\@tmpstrctr}{10}%
820 \let \@fc@ordstr#2\relax
821 \ifnum \@tmpstrctr=0\relax
822 \edef#2{\@fc@ordstr\@tenthstring{\@strctr}}%
823 \else
824 \edef#2{\@fc@ordstr\@tenstring{\@strctr}}%
825 \fi
826 \@strctr=#1\relax \@FCmodulo{\@strctr}{10}%
827 \ifnum \@strctr>0\relax
828 \let \@fc@ordstr#2\relax
829 \edef#2{\@fc@ordstr-\@unitthstring{\@strctr}}%
830 \fi
831 \else
832 \ifnum \@strctr<10\relax
833 \ifnum \@strctr=0\relax
834 \ifnum#1<100\relax
835 \let \@fc@ordstr#2\relax
836 \edef#2{\@fc@ordstr\@unitthstring{\@strctr}}%
837 \fi
838 \else
839 \let \@fc@ordstr#2\relax
840 \edef#2{\@fc@ordstr\@unitthstring{\@strctr}}%
841 \fi
842 \else
843 \@FCmodulo{\@strctr}{10}%
844 \let \@fc@ordstr#2\relax
845 \edef#2{\@fc@ordstr\@teenthstring{\@strctr}}%
846 \fi
847 \fi
848 }%
849 \global\let \@@ordinalstringenglish\@ordinalstringenglish

```

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

```

850 \newcommand*{\@ordinalstringMenglish}[2]{%
851 \let \@unitthstring=\@unitthstringenglish
852 \let \@teenthstring=\@teenthstringenglish
853 \let \@tenthstring=\@tenthstringenglish
854 \let \@unitstring=\@unitstringenglish
855 \let \@teenstring=\@teenstringenglish
856 \let \@tenstring=\@tenstringenglish
857 \def \@andname{and}%
858 \def \@hundred{hundred}\def \@thousand{thousand}%
859 \def \@hundredth{hundredth}\def \@thousandth{thousandth}%
860 \@ordinalstringenglish{#1}{#2}%
861 }%

```

```

862 \global\let\@OrdinalstringMenglish\@OrdinalstringMenglish
  No gender in English, so make feminine and neuter same as masculine:
863 \global\let\@OrdinalstringFenglish=\@OrdinalstringMenglish
864 \global\let\@OrdinalstringNenglish=\@OrdinalstringMenglish
  First letter of each word in upper case:
865 \newcommand*{\@OrdinalstringMenglish}[2]{%
866   \let\@unitthstring=\@Unitthstringenglish
867   \let\@teenthstring=\@Teenthstringenglish
868   \let\@tenthstring=\@Tenthstringenglish
869   \let\@unitstring=\@Unitstringenglish
870   \let\@teenstring=\@Teenstringenglish
871   \let\@tenstring=\@Tenstringenglish
872   \def\@andname{and}%
873   \def\@hundred{Hundred}\def\@thousand{Thousand}%
874   \def\@hundredth{Hundredth}\def\@thousandth{Thousandth}%
875   \@Ordinalstringenglish{#1}{#2}%
876 }%
877 \global\let\@OrdinalstringMenglish\@OrdinalstringMenglish
  No gender in English, so make feminine and neuter same as masculine:
878 \global\let\@OrdinalstringFenglish=\@OrdinalstringMenglish
879 \global\let\@OrdinalstringNenglish=\@OrdinalstringMenglish

```

10.1.6 fc-francais.def

```

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

  Set |francais| to be equivalent to |french|.
882 \global\let\@ordinalMfrancais=\@ordinalMfrench
883 \global\let\@ordinalFfrancais=\@ordinalFfrench
884 \global\let\@ordinalNfrancais=\@ordinalNfrench
885 \global\let\@numberstringMfrancais=\@numberstringMfrench
886 \global\let\@numberstringFfrancais=\@numberstringFfrench
887 \global\let\@numberstringNfrancais=\@numberstringNfrench
888 \global\let\@NumberstringMfrancais=\@NumberstringMfrench
889 \global\let\@NumberstringFfrancais=\@NumberstringFfrench
890 \global\let\@NumberstringNfrancais=\@NumberstringNfrench
891 \global\let\@ordinalstringMfrancais=\@ordinalstringMfrench
892 \global\let\@ordinalstringFfrancais=\@ordinalstringFfrench
893 \global\let\@ordinalstringNfrancais=\@ordinalstringNfrench
894 \global\let\@OrdinalstringMfrancais=\@OrdinalstringMfrench
895 \global\let\@OrdinalstringFfrancais=\@OrdinalstringFfrench
896 \global\let\@OrdinalstringNfrancais=\@OrdinalstringNfrench

```

10.1.7 fc-french.def

Definitions for French.

```

897 \ProvidesFCLanguage{french}[2017/06/15]%

```

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

First of all we define two macros `\fc@gl@let` and `\fc@gl@def` used in place of `\let` and `\def` within options setting macros. This way we can control from outside these macros whether the respective `\let` or `\def` is group-local or global. By default they are defined to be group-local.

```
898 \ifcsundef{fc@gl@let}{\global\let\fc@gl@let\let}{\PackageError{fmtcount}{Command already defined}}
899 \protect\fc@gl@let\space already defined.}}
900 \ifcsundef{fc@gl@def}{\global\let\fc@gl@def\def}{\PackageError{fmtcount}{Command already defined}}
901 \protect\fc@gl@def\space already defined.}}
```

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'.
902 \gdef\fc@french@set@plural#1#2#3#4#5#6{%
903   \ifthenelse{\equal{#2}{reformed}}{%
904     \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{#3}%
905   }{%
906     \ifthenelse{\equal{#2}{traditional}}{%
907       \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{#4}%
908     }{%
909       \ifthenelse{\equal{#2}{reformed o}}{%
910         \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{#5}%
911       }{%
912         \ifthenelse{\equal{#2}{traditional o}}{%
913           \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{#6}%
914         }{%
915           \ifthenelse{\equal{#2}{always}}{%
916             \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{0}%
917           }{%
918             \ifthenelse{\equal{#2}{never}}{%
919               \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{1}%
920             }{%
921               \ifthenelse{\equal{#2}{multiple}}{%
922                 \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{2}%
923               }{%
924                 \ifthenelse{\equal{#2}{multiple g-last}}{%
925                   \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{3}%
926                 }{%
927                   \ifthenelse{\equal{#2}{multiple l-last}}{%
928                     \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{4}%
929                   }{%
```


traditional use dash for numbers below 100, except when ‘et’ is used, and space otherwise
 reformed reform of 1990, use dash except with million & milliard, and suchlikes, i.e. $\langle n \rangle$ illion and $\langle n \rangle$ illiard,
 always always use dashes to separate all words

```

963 \define@key{fcfrench}{dash or space}[reformed]{%
964   \ifthenelse{\equal{#1}{traditional}}{%
965     \let\fc@frenchoptions@supermillion@dos\space%
966     \let\fc@frenchoptions@submillion@dos\space
967   }{%
968     \ifthenelse{\equal{#1}{reformed}\or\equal{#1}{1990}}{%
969       \let\fc@frenchoptions@supermillion@dos\space
970       \def\fc@frenchoptions@submillion@dos{-}%
971     }{%
972       \ifthenelse{\equal{#1}{always}}{%
973         \def\fc@frenchoptions@supermillion@dos{-}%
974         \def\fc@frenchoptions@submillion@dos{-}%
975       }{%
976         \PackageError{fmtcount}{Unexpected argument}{%
977           French option ‘dash or space’ expects ‘always’, ‘reformed’ or ‘traditional’
978         }
979       }%
980     }%
981   }%
982 }%

```

Option ‘scale’, can take 3 possible values:

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

```

983 \define@key{fcfrench}{scale}[recursive]{%
984   \ifthenelse{\equal{#1}{long}}{%
985     \let\fc@poweroften\fc@@pot@longscalefrench
986   }{%
987     \ifthenelse{\equal{#1}{recursive}}{%
988       \let\fc@poweroften\fc@@pot@recursivefrench
989     }{%
990       \ifthenelse{\equal{#1}{short}}{%
991         \let\fc@poweroften\fc@@pot@shortscalefrench
992       }{%
993         \PackageError{fmtcount}{Unexpected argument}{%
994           French option ‘scale’ expects ‘long’, ‘recursive’ or ‘short’
995         }
996       }%
997     }%
998   }%
999 }%

```


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

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

```

1000 \define@key{fcfrench}{n-illiard upto}[infinity]{%
1001   \ifthenelse{\equal{#1}{infinity}}{%
1002     \def\fc@longscale@nilliard@upto{0}%
1003   }{%
1004     \ifthenelse{\equal{#1}{infy}}{%
1005       \def\fc@longscale@nilliard@upto{0}%
1006     }{%
1007       \if Q\ifnum9<1#1Q\fi\else
1008         \PackageError{fmtcount}{Unexpected argument}{%
1009           French option ‘milliard threshold’ expects ‘infinity’, or equivalently ‘infy’, or a no
1010           integer.}%
1011         \fi
1012         \def\fc@longscale@nilliard@upto{#1}%
1013       }%
1014 }%

```

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.

```

1015 \def\@tempa#1{%
1016   \define@key{fcfrench}{#1}[]{%
1017     \PackageError{fmtcount}{Unexpected argument}{French option with key ‘#1’ does not take
1018     any value}}%
1019   \csgdef{KV@fcfrench@#1@default}{%
1020     \fc@gl@def\fmtcount@french{#1}}%
1021 }%
1022 \@tempa{france}\@tempa{swiss}\@tempa{belgian}%

```

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

```

1023 \gdef\fmtcount@french{france}%

```

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

```

1024 \define@key{fcfrench}{dialect}[france]{%
1025   \ifthenelse{\equal{#1}{france}
1026     \or\equal{#1}{swiss}
1027     \or\equal{#1}{belgian}}{%
1028     \def\fmtcount@french{#1}}{%
1029     \PackageError{fmtcount}{Invalid value ‘#1’ to french option dialect key}
1030     {Option ‘french’ can only take the values ‘france’,
1031     ‘belgian’ or ‘swiss’}}%
1032 \expandafter\@tempb\csname KV@fcfrench@dialect\endcsname

```

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’.

```

1033 \define@key{fcfrench}{mil plural mark}[le]{%
1034   \def\fc@frenchoptions@mil@plural@mark{#1}}
1035 \expandafter\@tempb\csname KV@fcfrench@mil plural mark\endcsname

```

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

The macro `\fc@UpperCaseFirstLetter` is such that `\fc@UpperCaseFirstLetter<word>\@nil` expands to `\word` with first letter capitalized and remainder unchanged.

```

1036 \gdef\fc@UpperCaseFirstLetter#1#2\@nil{%
1037   \uppercase{#1}#2}

```

The macro `\fc@CaseIden` is such that `\fc@CaseIden<word>\@nil` expands to `\word` unchanged.

```

1038 \gdef\fc@CaseIden#1\@nil{%
1039   #1%
1040 }%

```

The macro `\fc@UpperCaseAll` is such that `\fc@UpperCaseAll<word>\@nil` expands to `\word` all capitalized.

```

1041 \gdef\fc@UpperCaseAll#1\@nil{%
1042   \uppercase{#1}%
1043 }%

```

The macro `\fc@wcase` is the capitalizing macro for word-by-word capitalization. By default we set it to identity, ie. no capitalization.

```

1044 \global\let\fc@wcase\fc@CaseIden

```

The macro `\fc@gcase` is the capitalizing macro for global (the completed number) capitalization. By default we set it to identity, ie. no capitalization.

```

1045 \global\let\fc@gcase\fc@CaseIden

```

The macro `\fc@apply@gcase` simply applies `\fc@gcase` to `\@tempa`, knowing that `\@tempa` is the macro containing the result of formatting.

```

1046 \gdef\fc@apply@gcase{%

```

First of all we expand whatever `\fc@wcase... \@nil` found within `\@tempa`.

```

1047   \protected@edef\@tempa{\@tempa}%
1048   \protected@edef\@tempa{\expandafter\fc@gcase\@tempa\@nil}%
1049 }

```

`\ordinalMfrench`

```

1050 \newcommand*{\@ordinalMfrench}[2]{%
1051   \iffmtord@abbrv
1052     \ifnum#1=1 %
1053       \edef#2{\number#1\relax\noexpand\fmtord{er}}%
1054     \else
1055       \edef#2{\number#1\relax\noexpand\fmtord{e}}%
1056     \fi
1057 \else
1058   \PackageWarning{fmtcount}{Non abbreviated ordinal finals ('eme) are
1059     considered incorrect in French.}%

```

```

1060 \ifnum#1=1 %
1061   \edef#2{\number#1\relax\noexpand\fmtord{er}}%
1062 \else
1063   \protected@edef#2{\number#1\relax\noexpand\fmtord{\protect\'eme}}%
1064 \fi
1065 \fi}
1066 \global\let\@OrdinalMfrench\@OrdinalMfrench

```

@OrdinalFfrench

```

1067 \newcommand*{\@OrdinalFfrench}[2]{%
1068 \iffmtord@abbrv
1069 \ifnum#1=1 %
1070   \edef#2{\number#1\relax\noexpand\fmtord{re}}%
1071 \else
1072   \edef#2{\number#1\relax\noexpand\fmtord{e}}%
1073 \fi
1074 \else
1075 \PackageWarning{fmtcount}{Non abbreviated ordinal finals ('eme) are
1076   considered incorrect in French.}%
1077 \ifnum#1=1 %
1078   \protected@edef#2{\number#1\relax\noexpand\fmtord{\protect\'ere}}%
1079 \else
1080   \protected@edef#2{\number#1\relax\noexpand\fmtord{\protect\'eme}}%
1081 \fi
1082 \fi}
1083 \global\let\@OrdinalFfrench\@OrdinalFfrench

```

In French neutral gender and masculine gender are formally identical.

```

1084 \global\let\@OrdinalNfrench\@OrdinalMfrench

```

unitstringfrench

```

1085 \newcommand*{\@Unitstringfrench}[1]{%
1086 \noexpand\fc@wcase
1087 \ifcase#1 %
1088 z\'ero%
1089 \or un%
1090 \or deux%
1091 \or trois%
1092 \or quatre%
1093 \or cinq%
1094 \or six%
1095 \or sept%
1096 \or huit%
1097 \or neuf%
1098 \fi
1099 \noexpand\@nil
1100 }%
1101 \global\let\@Unitstringfrench\@Unitstringfrench

```

tenstringfrench

```

1102 \newcommand*{\@Tenstringfrench}[1]{%

```

```

1103 \noexpand\fc@wcase
1104 \ifcase#1 %
1105 \or dix%
1106 \or vingt%
1107 \or trente%
1108 \or quarante%
1109 \or cinquante%
1110 \or soixante%
1111 \or septante%
1112 \or huitante%
1113 \or nonante%
1114 \or cent%
1115 \fi
1116 \noexpand\@nil
1117 }%
1118 \global\let\@@@tenstringfrench\@@@tenstringfrench

```

teenstringfrench

```

1119 \newcommand*\@@@teenstringfrench}[1]{%
1120 \noexpand\fc@wcase
1121 \ifcase#1 %
1122 \or dix%
1123 \or onze%
1124 \or douze%
1125 \or treize%
1126 \or quatorze%
1127 \or quinze%
1128 \or seize%
1129 \or dix\noexpand\@nil-\noexpand\fc@wcase sept%
1130 \or dix\noexpand\@nil-\noexpand\fc@wcase huit%
1131 \or dix\noexpand\@nil-\noexpand\fc@wcase neuf%
1132 \fi
1133 \noexpand\@nil
1134 }%
1135 \global\let\@@@teenstringfrench\@@@teenstringfrench

```

seventiesfrench

```

1136 \newcommand*\@@@seventiesfrench}[1]{%
1137 \@tenstring{6}%
1138 \ifnum#1=1 %
1139 \fc@frenchoptions@submillion@dos\@andname\fc@frenchoptions@submillion@dos
1140 \else
1141 -%
1142 \fi
1143 \@teenstring{#1}%
1144 }%
1145 \global\let\@@@seventiesfrench\@@@seventiesfrench

```

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".

```
1146 \newcommand*{\@@eightiesfrench[1]{%
1147 \fc@wcase quatre\@nil-\noexpand\fc@wcase vingt%
1148 \ifnum#1>0 %
1149 \ifnum\fc@frenchoptions@vingt@plural=0 % vingt plural=always
1150 s%
1151 \fi
1152 \noexpand\@nil
1153 -\@unitstring{#1}%
1154 \else
1155 \ifcase\fc@frenchoptions@vingt@plural\space
1156 s% 0: always
1157 \or
1158 % 1: never
1159 \or
1160 s% 2: multiple
1161 \or
1162 % 3: multiple g-last
1163 \ifnum\count0=\count6\ifnum\count9=0 s\fi\fi
1164 \or
1165 % 4: multiple l-last
1166 \ifnum\count9=1 %
1167 \else
1168 s%
1169 \fi
1170 \or
1171 % 5: multiple lng-last
1172 \ifnum\count9=1 %
1173 \else
1174 \ifnum\count0>0 %
1175 s%
1176 \fi
1177 \fi
1178 \or
1179 % or 6: multiple ng-last
1180 \ifnum\count0>0 %
1181 s%
1182 \fi
1183 \fi
1184 \noexpand\@nil
1185 \fi
1186 }%
```

```

1187 \global\let\@@eightiesfrench\@@eightiesfrench
1188 \newcommand*\@@ninetiesfrench}[1]{%
1189 \fc@wcase quatre\@nil-\noexpand\fc@wcase vingt%
1190 \ifnum\fc@frenchoptions@vingt@plural=0 % vingt plural=always
1191 s%
1192 \fi
1193 \noexpand\@nil
1194 -\@teenstring{#1}%
1195 }%
1196 \global\let\@@ninetiesfrench\@@ninetiesfrench
1197 \newcommand*\@@seventiesfrenchswiss}[1]{%
1198 \@tenstring{7}%
1199 \ifnum#1=1\ \@andname\ \fi
1200 \ifnum#1>1-\fi
1201 \ifnum#1>0 \@unitstring{#1}\fi
1202 }%
1203 \global\let\@@seventiesfrenchswiss\@@seventiesfrenchswiss
1204 \newcommand*\@@eightiesfrenchswiss}[1]{%
1205 \@tenstring{8}%
1206 \ifnum#1=1\ \@andname\ \fi
1207 \ifnum#1>1-\fi
1208 \ifnum#1>0 \@unitstring{#1}\fi
1209 }%
1210 \global\let\@@eightiesfrenchswiss\@@eightiesfrenchswiss
1211 \newcommand*\@@ninetiesfrenchswiss}[1]{%
1212 \@tenstring{9}%
1213 \ifnum#1=1\ \@andname\ \fi
1214 \ifnum#1>1-\fi
1215 \ifnum#1>0 \@unitstring{#1}\fi
1216 }%
1217 \global\let\@@ninetiesfrenchswiss\@@ninetiesfrenchswiss

```

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

```

1218 \newcommand*\fc@french@common{%
1219 \let\fc@wcase\fc@CaseIden
1220 \let\@unitstring=\@unitstringfrench
1221 \let\@teenstring=\@teenstringfrench
1222 \let\@tenstring=\@tenstringfrench
1223 \def\@hundred{cent}%
1224 \def\@andname{et}%
1225 }%
1226 \global\let\fc@french@common\fc@french@common
1227 \newcommand*\@numberstringMfrenchswiss}[2]{%
1228 \fc@french@common
1229 \let\fc@gcase\fc@CaseIden
1230 \let\@seventies=\@@seventiesfrenchswiss
1231 \let\@eighties=\@@eightiesfrenchswiss
1232 \let\@nineties=\@@ninetiesfrenchswiss

```

```

1233 \let\fc@nbrstr@preamble@empty
1234 \let\fc@nbrstr@postamble@empty
1235 \@@numberstringfrench{#1}{#2}}
1236 \global\let\@numberstringMfrenchswiss\@numberstringMfrenchswiss
1237 \newcommand*{\@numberstringMfrenchfrance}[2]{%
1238 \fc@french@common
1239 \let\fc@gcase\fc@CaseIden
1240 \let\@seventies=\@@seventiesfrench
1241 \let\@eighties=\@@eightiesfrench
1242 \let\@nineties=\@@ninetiesfrench
1243 \let\fc@nbrstr@preamble@empty
1244 \let\fc@nbrstr@postamble@empty
1245 \@@numberstringfrench{#1}{#2}}
1246 \global\let\@numberstringMfrenchfrance\@numberstringMfrenchfrance
1247 \newcommand*{\@numberstringMfrenchbelgian}[2]{%
1248 \fc@french@common
1249 \let\fc@gcase\fc@CaseIden
1250 \let\@seventies=\@@seventiesfrenchswiss
1251 \let\@eighties=\@@eightiesfrench
1252 \let\@nineties=\@@ninetiesfrench
1253 \let\fc@nbrstr@preamble@empty
1254 \let\fc@nbrstr@postamble@empty
1255 \@@numberstringfrench{#1}{#2}}
1256 \global\let\@numberstringMfrenchbelgian\@numberstringMfrenchbelgian
1257 \let\@numberstringMfrench=\@numberstringMfrenchfrance
1258 \newcommand*{\@numberstringFfrenchswiss}[2]{%
1259 \fc@french@common
1260 \let\fc@gcase\fc@CaseIden
1261 \let\@seventies=\@@seventiesfrenchswiss
1262 \let\@eighties=\@@eightiesfrenchswiss
1263 \let\@nineties=\@@ninetiesfrenchswiss
1264 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
1265 \let\fc@nbrstr@postamble@empty
1266 \@@numberstringfrench{#1}{#2}}
1267 \global\let\@numberstringFfrenchswiss\@numberstringFfrenchswiss
1268 \newcommand*{\@numberstringFfrenchfrance}[2]{%
1269 \fc@french@common
1270 \let\fc@gcase\fc@CaseIden
1271 \let\@seventies=\@@seventiesfrench
1272 \let\@eighties=\@@eightiesfrench
1273 \let\@nineties=\@@ninetiesfrench
1274 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
1275 \let\fc@nbrstr@postamble@empty
1276 \@@numberstringfrench{#1}{#2}}
1277 \global\let\@numberstringFfrenchfrance\@numberstringFfrenchfrance
1278 \newcommand*{\@numberstringFfrenchbelgian}[2]{%
1279 \fc@french@common
1280 \let\fc@gcase\fc@CaseIden
1281 \let\@seventies=\@@seventiesfrenchswiss

```

```

1282 \let\@eighties=\@eightiesfrench
1283 \let\@nineties=\@ninetiesfrench
1284 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
1285 \let\fc@nbrstr@postamble\@empty
1286 \@@numberstringfrench{#1}{#2}}
1287 \global\let\@NumberstringFfrenchbelgian\@NumberstringFfrenchbelgian
1288 \global\let\@NumberstringFfrench=\@NumberstringFfrenchfrance
1289 \global\let\@OrdinalstringNfrench\@OrdinalstringMfrench
1290 \newcommand*{\@NumberstringMfrenchswiss}[2]{%
1291 \fc@french@common
1292 \let\fc@gcase\fc@UpperCaseFirstLetter
1293 \let\@seventies=\@seventiesfrenchswiss
1294 \let\@eighties=\@eightiesfrenchswiss
1295 \let\@nineties=\@ninetiesfrenchswiss
1296 \let\fc@nbrstr@preamble\@empty
1297 \let\fc@nbrstr@postamble\fc@apply@gcase
1298 \@@numberstringfrench{#1}{#2}}
1299 \global\let\@NumberstringMfrenchswiss\@NumberstringMfrenchswiss
1300 \newcommand*{\@NumberstringMfrenchfrance}[2]{%
1301 \fc@french@common
1302 \let\fc@gcase\fc@UpperCaseFirstLetter
1303 \let\@seventies=\@seventiesfrench
1304 \let\@eighties=\@eightiesfrench
1305 \let\@nineties=\@ninetiesfrench
1306 \let\fc@nbrstr@preamble\@empty
1307 \let\fc@nbrstr@postamble\fc@apply@gcase
1308 \@@numberstringfrench{#1}{#2}}
1309 \global\let\@NumberstringMfrenchfrance\@NumberstringMfrenchfrance
1310 \newcommand*{\@NumberstringMfrenchbelgian}[2]{%
1311 \fc@french@common
1312 \let\fc@gcase\fc@UpperCaseFirstLetter
1313 \let\@seventies=\@seventiesfrenchswiss
1314 \let\@eighties=\@eightiesfrench
1315 \let\@nineties=\@ninetiesfrench
1316 \let\fc@nbrstr@preamble\@empty
1317 \let\fc@nbrstr@postamble\fc@apply@gcase
1318 \@@numberstringfrench{#1}{#2}}
1319 \global\let\@NumberstringMfrenchbelgian\@NumberstringMfrenchbelgian
1320 \global\let\@NumberstringMfrench=\@NumberstringMfrenchfrance
1321 \newcommand*{\@NumberstringFfrenchswiss}[2]{%
1322 \fc@french@common
1323 \let\fc@gcase\fc@UpperCaseFirstLetter
1324 \let\@seventies=\@seventiesfrenchswiss
1325 \let\@eighties=\@eightiesfrenchswiss
1326 \let\@nineties=\@ninetiesfrenchswiss
1327 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
1328 \let\fc@nbrstr@postamble\fc@apply@gcase
1329 \@@numberstringfrench{#1}{#2}}
1330 \global\let\@NumberstringFfrenchswiss\@NumberstringFfrenchswiss

```



```

1331 \newcommand*{\@NumberstringFfrenchfrance}[2]{%
1332 \fc@french@common
1333 \let\fc@gcase\fc@UpperCaseFirstLetter
1334 \let\@seventies=\@@seventiesfrench
1335 \let\@eighties=\@@eightiesfrench
1336 \let\@nineties=\@@ninetiesfrench
1337 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
1338 \let\fc@nbrstr@postamble\fc@apply@gcase
1339 \@@numberstringfrench{#1}{#2}}
1340 \global\let\@NumberstringFfrenchfrance\@NumberstringFfrenchfrance
1341 \newcommand*{\@NumberstringFfrenchbelgian}[2]{%
1342 \fc@french@common
1343 \let\fc@gcase\fc@UpperCaseFirstLetter
1344 \let\@seventies=\@@seventiesfrenchswiss
1345 \let\@eighties=\@@eightiesfrench
1346 \let\@nineties=\@@ninetiesfrench
1347 \let\fc@nbrstr@preamble\fc@@nbrstr@Fpreamble
1348 \let\fc@nbrstr@postamble\fc@apply@gcase
1349 \@@numberstringfrench{#1}{#2}}
1350 \global\let\@NumberstringFfrenchbelgian\@NumberstringFfrenchbelgian
1351 \global\let\@NumberstringFfrench=\@NumberstringFfrenchfrance
1352 \global\let\@NumberstringNfrench\@NumberstringMfrench
1353 \newcommand*{\@ordinalstringMfrenchswiss}[2]{%
1354 \fc@french@common
1355 \let\fc@gcase\fc@CaseIden
1356 \let\fc@first\fc@@firstfrench
1357 \let\@seventies=\@@seventiesfrenchswiss
1358 \let\@eighties=\@@eightiesfrenchswiss
1359 \let\@nineties=\@@ninetiesfrenchswiss
1360 \@@ordinalstringfrench{#1}{#2}%
1361 }%
1362 \global\let\@ordinalstringMfrenchswiss\@ordinalstringMfrenchswiss
1363 \newcommand*\fc@@firstfrench{premier}
1364 \global\let\fc@@firstfrench\fc@@firstfrench

1365 \newcommand*\fc@@firstFfrench{premi\protect\`ere}
1366 \global\let\fc@@firstFfrench\fc@@firstFfrench
1367 \newcommand*{\@ordinalstringMfrenchfrance}[2]{%
1368 \fc@french@common
1369 \let\fc@gcase\fc@CaseIden
1370 \let\fc@first=\fc@@firstfrench
1371 \let\@seventies=\@@seventiesfrench
1372 \let\@eighties=\@@eightiesfrench
1373 \let\@nineties=\@@ninetiesfrench
1374 \@@ordinalstringfrench{#1}{#2}}
1375 \global\let\@ordinalstringMfrenchfrance\@ordinalstringMfrenchfrance
1376 \newcommand*{\@ordinalstringMfrenchbelgian}[2]{%
1377 \fc@french@common
1378 \let\fc@gcase\fc@CaseIden
1379 \let\fc@first=\fc@@firstfrench

```

```

1380 \let\@seventies=\@seventiesfrench
1381 \let\@eighties=\@eightiesfrench
1382 \let\@nineties=\@ninetiesfrench
1383 \@ordinalstringfrench{#1}{#2}%
1384 }%
1385 \global\let\@ordinalstringMfrenchbelgian\@ordinalstringMfrenchbelgian
1386 \global\let\@ordinalstringMfrench=\@ordinalstringMfrenchfrance
1387 \newcommand*\@ordinalstringFfrenchswiss}[2]{%
1388 \fc@french@common
1389 \let\fc@gcase\fc@CaseIden
1390 \let\fc@first\fc@@firstFfrench
1391 \let\@seventies=\@seventiesfrenchswiss
1392 \let\@eighties=\@eightiesfrenchswiss
1393 \let\@nineties=\@ninetiesfrenchswiss
1394 \@ordinalstringfrench{#1}{#2}%
1395 }%
1396 \global\let\@ordinalstringFfrenchswiss\@ordinalstringFfrenchswiss
1397 \newcommand*\@ordinalstringFfrenchfrance}[2]{%
1398 \fc@french@common
1399 \let\fc@gcase\fc@CaseIden
1400 \let\fc@first=\fc@@firstFfrench
1401 \let\@seventies=\@seventiesfrench
1402 \let\@eighties=\@eightiesfrench
1403 \let\@nineties=\@ninetiesfrench
1404 \@ordinalstringfrench{#1}{#2}%
1405 }%
1406 \global\let\@ordinalstringFfrenchfrance\@ordinalstringFfrenchfrance
1407 \newcommand*\@ordinalstringFfrenchbelgian}[2]{%
1408 \fc@french@common
1409 \let\fc@gcase\fc@CaseIden
1410 \let\fc@first=\fc@@firstFfrench
1411 \let\@seventies=\@seventiesfrench
1412 \let\@eighties=\@eightiesfrench
1413 \let\@nineties=\@ninetiesfrench
1414 \@ordinalstringfrench{#1}{#2}%
1415 }%
1416 \global\let\@ordinalstringFfrenchbelgian\@ordinalstringFfrenchbelgian
1417 \global\let\@ordinalstringFfrench=\@ordinalstringFfrenchfrance
1418 \global\let\@ordinalstringNfrench\@ordinalstringMfrench
1419 \newcommand*\@OrdinalstringMfrenchswiss}[2]{%
1420 \fc@french@common
1421 \let\fc@gcase\fc@UpperCaseFirstLetter
1422 \let\fc@first=\fc@@firstfrench
1423 \let\@seventies=\@seventiesfrenchswiss
1424 \let\@eighties=\@eightiesfrenchswiss
1425 \let\@nineties=\@ninetiesfrenchswiss
1426 \@ordinalstringfrench{#1}{#2}%
1427 }%
1428 \global\let\@OrdinalstringMfrenchswiss\@OrdinalstringMfrenchswiss

```

```

1429 \newcommand*{\@OrdinalstringMfrenchfrance}[2]{%
1430 \fc@french@common
1431 \let\fc@gcase\fc@UpperCaseFirstLetter
1432 \let\fc@first\fc@@firstfrench
1433 \let\@seventies=\@seventiesfrench
1434 \let\@eighties=\@eightiesfrench
1435 \let\@nineties=\@ninetiesfrench
1436 \@ordinalstringfrench{#1}{#2}%
1437 }%
1438 \global\let\@OrdinalstringMfrenchfrance\@OrdinalstringMfrenchfrance
1439 \newcommand*{\@OrdinalstringMfrenchbelgian}[2]{%
1440 \fc@french@common
1441 \let\fc@gcase\fc@UpperCaseFirstLetter
1442 \let\fc@first\fc@@firstfrench
1443 \let\@seventies=\@seventiesfrench
1444 \let\@eighties=\@eightiesfrench
1445 \let\@nineties=\@ninetiesfrench
1446 \@ordinalstringfrench{#1}{#2}%
1447 }%
1448 \global\let\@OrdinalstringMfrenchbelgian\@OrdinalstringMfrenchbelgian
1449 \global\let\@OrdinalstringMfrench=\@OrdinalstringMfrenchfrance
1450 \newcommand*{\@OrdinalstringFfrenchswiss}[2]{%
1451 \fc@french@common
1452 \let\fc@gcase\fc@UpperCaseFirstLetter
1453 \let\fc@first\fc@@firstfrench
1454 \let\@seventies=\@seventiesfrenchswiss
1455 \let\@eighties=\@eightiesfrenchswiss
1456 \let\@nineties=\@ninetiesfrenchswiss
1457 \@ordinalstringfrench{#1}{#2}%
1458 }%
1459 \global\let\@OrdinalstringFfrenchswiss\@OrdinalstringFfrenchswiss
1460 \newcommand*{\@OrdinalstringFfrenchfrance}[2]{%
1461 \fc@french@common
1462 \let\fc@gcase\fc@UpperCaseFirstLetter
1463 \let\fc@first\fc@@firstFfrench
1464 \let\@seventies=\@seventiesfrench
1465 \let\@eighties=\@eightiesfrench
1466 \let\@nineties=\@ninetiesfrench
1467 \@ordinalstringfrench{#1}{#2}%
1468 }%
1469 \global\let\@OrdinalstringFfrenchfrance\@OrdinalstringFfrenchfrance
1470 \newcommand*{\@OrdinalstringFfrenchbelgian}[2]{%
1471 \fc@french@common
1472 \let\fc@gcase\fc@UpperCaseFirstLetter
1473 \let\fc@first\fc@@firstFfrench
1474 \let\@seventies=\@seventiesfrench
1475 \let\@eighties=\@eightiesfrench
1476 \let\@nineties=\@ninetiesfrench
1477 \@ordinalstringfrench{#1}{#2}%

```

```

1478 }%
1479 \global\let\@OrdinalstringFfrenchbelgian\@OrdinalstringFfrenchbelgian
1480 \global\let\@OrdinalstringFfrench=\@OrdinalstringFfrenchfrance
1481 \global\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.

```

1482 \ifcsundef{fc@@do@plural@mark}{}%
1483 {\PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1484   'fc@@do@plural@mark'}}

```

Arguments as follows:

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

Implicit arguments as follows:

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

```

1485 \def\fc@@do@plural@mark#1{%
1486   \ifcase\count10 %
1487     #1% 0=always
1488     \or% 1=never
1489     \or% 2=multiple
1490     \ifnum\count1>1 %
1491       #1%
1492       \fi
1493     \or% 3= multiple g-last
1494     \ifnum\count1>1 %
1495       \ifnum\count0=\count6 %
1496         #1%
1497         \fi
1498       \fi
1499     \or% 4= multiple l-last
1500     \ifnum\count1>1 %
1501       \ifnum\count9=1 %
1502         \else
1503         #1%
1504         \fi
1505       \fi
1506     \or% 5= multiple lng-last
1507     \ifnum\count1>1 %
1508       \ifnum\count9=1 %
1509       \else
1510       \if\count0>\count6 %
1511       #1%
1512       \fi

```


#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

```
1550 \def\fc@pot@longscalefrench#1#2#3{%
1551   {%
```

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

```
1552   \edef\@tempb{\number#1}%
```

Let `\count1` be the plural value.

```
1553   \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 .

```
1554   \count2\count0 %
1555   \divide\count2 by 6 %
1556   \count3\count2 %
1557   \multiply\count3 by 6 %
1558   \count3-\count3 %
1559   \advance\count3 by \count0 %
1560   \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)”.

```
1561   \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 \@tempb` to ‘1’ for “mil(le)”, and to ‘2’ otherwise.

```
1562   \edef\@tempb{%
1563     \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.

```
1564     1%
1565     \else
1566     \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`.

```
1567     \ifnum\fc@longscale@nilliard@upto=0 %
```

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

```
1568     2%
1569     \else
```

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

```

1570         \ifnum\count2>\fc@longscale@nilliard@upto
1571             1%
1572         \else
1573             2%
1574         \fi
1575     \fi
1576 \else
1577     2%
1578 \fi
1579 \fi
1580 }%
1581 \ifnum\@temph=1 %

```

Here 10^w is formatted as “mil(le)”.

```

1582     \count10=\fc@frenchoptions@mil@plural\space
1583     \edef\@tempe{%
1584         \noexpand\fc@wcase
1585         mil%
1586         \fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark
1587         \noexpand\@nil
1588     }%
1589 \else
1590     % weight >= 6
1591     \expandafter\fc@@latin@cardinal@pefix\expandafter{\the\count2}\@tempg
1592     % now form the xxx-illion(s) or xxx-illiard(s) word
1593     \ifnum\count3>2 %
1594         \toks10{illiard}%
1595         \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space
1596     \else
1597         \toks10{illion}%
1598         \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
1599     \fi
1600     \edef\@tempe{%
1601         \noexpand\fc@wcase
1602         \@tempg
1603         \the\toks10 %
1604         \fc@@do@plural@mark s%
1605         \noexpand\@nil
1606     }%
1607     \fi
1608 \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.

```

1609     \let\@tempe\@empty
1610     \def\@temph{0}%
1611 \fi
1612 \else

```

Case of $w = 0$.

```
1613 \let\@tempe\@empty
1614 \def\@temph{0}%
1615 \fi
```

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

```
1616 \expandafter\toks\expandafter1\expandafter{\@tempe}%
1617 \toks0{#2}%
1618 \edef\@tempa{\the\toks0 \@temph \def\noexpand#3{\the\toks1}}%
1619 \expandafter
1620 }\@tempa
1621 }%
1622 \global\let\fc@pot@longscalefrench\fc@pot@longscalefrench
```

`\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.

```
1623 \ifcsundef\fc@pot@shortscalefrench\{}{%
1624 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1625 '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

```
1626 \def\fc@pot@shortscalefrench#1#2#3{%
1627 {%
```

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

```
1628 \edef\@tempb{\number#1}%
```

And let `\count1` be the plural value.

```
1629 \count1=\@tempb
```

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

```
1630 \count2\count0 %
1631 \divide\count2 by 3 %
1632 \advance\count2 by -1 %
```

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

```
1633 \ifnum\count0>0 % If weight>=3, i.e we do have to append thousand or n-illion(s)/n-illiard(s)
```



```

1634 \ifnum\count1>0 % we have at least one thousand/n-illion/n-illiard
1635 \ifnum\count2=0 %
1636 \def\@temph{1}%
1637 \count1=\fc@frenchoptions@mil@plural\space
1638 \edef\@tempe{%
1639 mil%
1640 \fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark
1641 }%
1642 \else
1643 \def\@temph{2}%
1644 % weight >= 6
1645 \expandafter\fc@@latin@cardinal@prefix\expandafter{\the\count2}\@tempg
1646 \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
1647 \edef\@tempe{%
1648 \noexpand\fc@wcase
1649 \@tempg
1650 illion%
1651 \fc@@do@plural@mark s%
1652 \noexpand\@nil
1653 }%
1654 \fi
1655 \else

```

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

```

1656 \def\@temph{0}%
1657 \let\@tempe\@empty
1658 \fi
1659 \else

```

Here $w = 0$.

```

1660 \def\@temph{0}%
1661 \let\@tempe\@empty
1662 \fi
1663 % now place into \@tempa the assignment of results \@temph and \@tempe to to \text
1664 % \texttt{\#3} for further propagation after closing brace.
1665 % \begin{macrocode}
1666 \expandafter\toks\expandafter1\expandafter{\@tempe}%
1667 \toks0{\#2}%
1668 \edef\@tempa{\the\toks0 \@temph \def\noexpand#3{\the\toks1}}%
1669 \expandafter
1670 }\@tempa
1671 }%
1672 \global\let\fc@pot@shortscalefrench\fc@pot@shortscalefrench

```

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.

```

1673 \ifcsundef{fc@pot@recursivefrench}{-}{%
1674 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1675 '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

```
1676 \def\fc@pot@recursivefrench#1#2#3{%
1677   {%
```

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

```
1678   \edef\@tempb{\number#1}%
1679   \let\@tempa\@tempa
```

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

```
1680   \count1=\@tempb\space
```

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

```
1681   \ifnum\count1>0 %
1682     \count2\count0 %
1683     \divide\count2 by 9 %
1684     \advance\count2 by -1 %
1685     \let\@tempe\@empty
1686     \edef\@tempf{\fc@frenchoptions@supermillion@dos
1687       de\fc@frenchoptions@supermillion@dos\fc@wcase milliards\@nil}%
1688     \count11\count0 %
1689     \ifnum\count2>0 %
1690       \count3\count2 %
1691       \count3-\count3 %
1692       \multiply\count3 by 9 %
1693       \advance\count11 by \count3 %
1694       \loop
1695         % (\count2, \count3) <- (\count2 div 2, \count2 mod 2)
1696         \count3\count2 %
1697         \divide\count3 by 2 %
1698         \multiply\count3 by 2 %
1699         \count3-\count3 %
1700         \advance\count3 by \count2 %
1701         \divide\count2 by 2 %
1702         \ifnum\count3=1 %
1703           \let\@tempg\@tempe
1704           \edef\@tempe{\@tempg\@tempf}%
1705         \fi
1706         \let\@tempg\@tempf
1707         \edef\@tempf{\@tempg\@tempg}%
1708         \ifnum\count2>0 %
1709           \repeat
1710         \fi
1711       \divide\count11 by 3 %
```

```

1712 \ifcase\count11 % 0 .. 5
1713     % 0 => d milliard(s) (de milliards)*
1714     \def\@temph{2}%
1715     \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space
1716 \or % 1 => d mille milliard(s) (de milliards)*
1717     \def\@temph{1}%
1718     \count10=\fc@frenchoptions@mil@plural\space
1719 \or % 2 => d million(s) (de milliards)*
1720     \def\@temph{2}%
1721     \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
1722 \or % 3 => d milliard(s) (de milliards)*
1723     \def\@temph{2}%
1724     \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space
1725 \or % 4 => d mille milliards (de milliards)*
1726     \def\@temph{1}%
1727     \count10=\fc@frenchoptions@mil@plural\space
1728 \else % 5 => d million(s) (de milliards)*
1729     \def\@temph{2}%
1730     \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
1731 \fi
1732 \let\@tempg\@tempe
1733 \edef\@tempf{%
1734     \ifcase\count11 % 0 .. 5
1735     \or
1736         mil\fc@@do@plural@mark \fc@frenchoptions@mil@plural@mark
1737     \or
1738         million\fc@@do@plural@mark s%
1739     \or
1740         milliard\fc@@do@plural@mark s%
1741     \or
1742         mil\fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark
1743         \noexpand\@nil\fc@frenchoptions@supermillion@dos
1744         \noexpand\fc@wcase milliards% 4
1745     \or
1746         million\fc@@do@plural@mark s%
1747         \noexpand\@nil\fc@frenchoptions@supermillion@dos
1748         de\fc@frenchoptions@supermillion@dos\noexpand\fc@wcase  milliards% 5
1749     \fi
1750 }%
1751 \edef\@tempe{%
1752     \ifx\@tempf\@empty\else
1753         \expandafter\fc@wcase\@tempf\@nil
1754     \fi
1755     \@tempg
1756 }%
1757 \else
1758     \def\@temph{0}%
1759     \let\@tempe\@empty
1760 \fi

```

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

```

1761 \expandafter\toks\expandafter1\expandafter{\@tempb}%
1762 \toks0{#2}%
1763 \edef\@tempa{\the\toks0 \@tempa \def\noexpand#3{\the\toks1}}%
1764 \expandafter
1765 }\@tempa
1766 }%
1767 \global\let\fc@pot@recursivefrench\fc@pot@recursivefrench

```

`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.

```

1768 \ifcsundef{fc@muladdfrench}{-}{%
1769 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1770 'fc@muladdfrench'}}

```

Arguments as follows:

#2 input, plural indicator for number d

#3 input, formatted number d

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

Implicit arguments from context:

`\@tempa` input, formatted number a

output, macro to which place the mul-add result

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

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

```

1771 \def\fc@muladdfrench#1#2#3{%
1772 {%

```

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

```

1773 \edef\@tempc{#1}%
1774 \edef\@tempd{#2}%
1775 \edef\@tempf{#3}%
1776 \let\@tempc\@tempc
1777 \let\@tempd\@tempd

```

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

```

1778 \ifnum\@tempc=1 %
1779 \ifnum\count9=1 %
1780 \let\@tempd\empty
1781 \fi
1782 \fi

```

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

```

1783 \edef\@tempg{%
1784   \@tempd
1785   \ifx\@tempd\@empty\else
1786     \ifx\@tempf\@empty\else
1787       \ifcase\count9 %
1788         \or
1789         \fc@frenchoptions@submillion@dos
1790       \or
1791         \fc@frenchoptions@supermillion@dos
1792       \fi
1793     \fi
1794   \fi
1795   \@tempf
1796 }%
```

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

```

1797 \edef\@temph{%
1798   \@tempa
1799   \ifx\@tempa\@empty\else
1800     \ifx\@tempg\@empty\else
1801       \ifcase\count8 %
1802         \or
1803         \fc@frenchoptions@submillion@dos
1804       \or
1805         \fc@frenchoptions@supermillion@dos
1806       \fi
1807     \fi
1808   \fi
1809   \@tempg
1810 }%
```

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

```

1811 \def\@tempb##1{\def\@tempa{\def\@tempa{##1}}}%
1812 \expandafter\@tempb\expandafter{\@temph}%
1813 \expandafter
1814 }\@tempa
1815 }%
1816 \global\let\fc@muladdfrench\fc@muladdfrench
```

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

```

1817 \ifcsundef{fc@lthundredstringfrench}{-}%
1818 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1819   ‘fc@lthundredstringfrench’}
```

The number to format is not passed as an argument to this macro, instead each digit of it is in a \fc@digit@<w> macro after this number has been parsed. So the only thing that

`\fc@lthundredstringfrench` needs to know $\langle w \rangle$ which is passed as `\count0` for the less significant digit.

#1 input/output macro to which append the result

Implicit input arguments as follows:

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

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

```
1820 \def\fc@lthundredstringfrench#1{%
```

```
1821  {%
```

First save arguments into local temporary macro.

```
1822  \let\@tempc#1%
```

Read units d_w to `\count1`.

```
1823  \fc@read@unit{\count1}{\count0}%
```

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

```
1824  \count3\count0 %
```

```
1825  \advance\count3 1 %
```

```
1826  \fc@read@unit{\count2}{\count3}%
```

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

```
1827  \edef\@tempa{%
```

```
1828    \@tempc
```

```
1829    \ifnum\count2>1 %
```

```
1830      % 20 .. 99
```

```
1831      \ifnum\count2>6 %
```

```
1832        % 70 .. 99
```

```
1833        \ifnum\count2<8 %
```

```
1834          % 70 .. 79
```

```
1835          \@seventies{\count1}%
```

```
1836        \else
```

```
1837          % 80..99
```

```
1838          \ifnum\count2<9 %
```

```
1839            % 80 .. 89
```

```
1840            \@eighties{\count1}%
```

```
1841          \else
```

```
1842            % 90 .. 99
```

```
1843            \@nineties{\count1}%
```

```
1844          \fi
```

```
1845        \fi
```

```
1846      \else
```

```
1847        % 20..69
```

```
1848        \@tenstring{\count2}%
```

```
1849        \ifnum\count1>0 %
```

```
1850          % x1 .. x0
```

```
1851          \ifnum\count1=1 %
```

```
1852            % x1
```

```
1853            \fc@frenchoptions@submillion@dos\@andname\fc@frenchoptions@submillion@dos
```

```
1854          \else
```

```
1855            % x2 .. x9
```

```

1856         -%
1857         \fi
1858         \@unitstring{\count1}%
1859     \fi
1860     \fi
1861 \else
1862     % 0 .. 19
1863     \ifnum\count2=0 % when tens = 0
1864     % 0 .. 9
1865     \ifnum\count1=0 % when units = 0
1866     % \count3=1 when #1 = 0, i.e. only for the unit of the top level number
1867     \ifnum\count3=1 %
1868     \ifnum\fc@max@weight=0 %
1869     \@unitstring{0}%
1870     \fi
1871     \fi
1872     \else
1873     % 1 .. 9
1874     \@unitstring{\count1}%
1875     \fi
1876     \else
1877     % 10 .. 19
1878     \@teenstring{\count1}%
1879     \fi
1880 \fi
1881 }%

```

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

```

1882 \def\@tempb##1{\def\@tempa{\def#1{##1}}}%
1883 \expandafter\@tempb\expandafter{\@tempa}%
1884 \expandafter
1885 }\@tempa
1886 }%
1887 \global\let\fc@lthundredstringfrench\fc@lthundredstringfrench

```

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

```

1888 \ifcsundef\fc@ltthousandstringfrench}{-}{%
1889 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1890 '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)

```

1891 \def\fc@ltthousandstringfrench#1{%
1892   {%

```

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

```
1893 \count4\count0 %
1894 \advance\count4 by 2 %
1895 \fc@read@unit{\count2 }{\count4 }%
```

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

```
1896 \advance\count4 by -1 %
1897 \count3\count4 %
1898 \advance\count3 by -1 %
1899 \fc@check@nonzeros{\count3 }{\count4 }\@tempa
```

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

```
1900 \edef\@temps{%
1901 \ifcase\fc@frenchoptions@cent@plural\space
1902 % 0 => always
1903 s%
1904 \or
1905 % 1 => never
1906 \or
1907 % 2 => multiple
1908 \ifnum\count2>1s\fi
1909 \or
1910 % 3 => multiple g-last
1911 \ifnum\count2>1 \ifnum\@tempa=0 \ifnum\count0=\count6s\fi\fi\fi
1912 \or
1913 % 4 => multiple l-last
1914 \ifnum\count2>1 \ifnum\@tempa=0 \ifnum\count9=0s\else\ifnum\count9=2s\fi\fi\fi\fi
1915 \fi
1916 }%
1917 % compute spacing after cent(s?) into \@tempb
1918 \expandafter\let\expandafter\@tempb
1919 \ifnum\@tempa>0 \fc@frenchoptions@submillion@dos\else\@empty\fi
1920 % now place into \@tempa the hundreds
1921 \edef\@tempa{%
1922 \ifnum\count2=0 %
1923 \else
1924 \ifnum\count2=1 %
1925 \expandafter\fc@wcase\@hundred\@nil
1926 \else
1927 \@unitstring{\count2}\fc@frenchoptions@submillion@dos
1928 \noexpand\fc@wcase\@hundred\@temps\noexpand\@nil
1929 \fi
1930 \@tempb
1931 \fi
1932 }%
1933 % now append to \@tempa the ten and unit
1934 \fc@lthundredstringfrench\@tempa
```

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

```
1935 \def\@tempb##1{\def\@tempa{\def#1{##1}}}%
```



```
1936 \expandafter\@tempb\expandafter{\@tempa}%
```

```
1937 \expandafter
```

```
1938 }\@tempa
```

```
1939 }%
```

```
1940 \global\let\fc@lthousandstringfrench\fc@lthousandstringfrench
```

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

```
1941 \ifcsundef{@@numberstringfrench}{}%
```

```
1942 \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

```
1943 \def\@@numberstringfrench#1#2{%
```

```
1944 {%
```

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

```
1945 \edef\@tempa{#1}%
```

```
1946 \expandafter\fc@number@parser\expandafter{\@tempa}%
```

```
1947 \ifnum\fc@min@weight<0 %
```

```
1948 \PackageError{fmtcount}{Out of range}%
```

```
1949 {This macro does not work with fractional numbers}%
```

```
1950 \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’.

```
1951 \edef\@tempa{\ifcase\fc@sign@case\space\or\fc@wcase plus\@nil\or\fc@wcase moins\@nil\fi}%
```

```
1952 \fc@nbrstr@preamble
```

```
1953 \fc@@nbrstrfrench@inner
```

```
1954 \fc@nbrstr@postamble
```

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

```
1955 \def\@tempb##1{\def\@tempa{\def#2{##1}}}%
```

```
1956 \expandafter\@tempb\expandafter{\@tempa}%
```

```
1957 \expandafter
```

```
1958 }\@tempa
```

```
1959 }%
```

```
1960 \global\let\@@numberstringfrench\@@numberstringfrench
```

@@nbrstrfrench@innerCommon 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.

```
1961 \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.

```
1962 \count0=\fc@max@weight
```

```
1963 \divide\count0 by 3 %
```

```
1964 \multiply\count0 by 3 %
```

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

```
1965 \fc@intpart@find@last{\count5 }%
1966 \count6\count5 %
1967 \divide\count6 3 %
1968 \multiply\count6 3 %
1969 \count8=0 %
1970 \loop
```

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

```
1971 \count1\count0 %
1972 \advance\count1 by 2 %
1973 \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`.

```
1974 \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`

```
1975 \fc@ltthousandstringfrench \@tempd
```

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

```
1976 \fc@muladdfrench \@tempt \@tempd \@tempb
```

Then iterate.

```
1977 \count8\count9 %
1978 \advance\count0 by -3 %
1979 \ifnum\count6>\count0 \else
1980 \repeat
1981 }%
1982 \global\let\fc@nbrstrfrench@inner\fc@nbrstrfrench@inner
```

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

```
1983 \ifcsundef{\@@ordinalstringfrench}{}%
1984 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1985 '@@ordinalstringfrench'}}
```

Arguments are as follows:

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

```
1986 \def\@@ordinalstringfrench#1#2{%
1987 {%
```

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

```
1988 \edef\@tempa{#1}%
1989 \expandafter\fc@number@parser\expandafter{\@tempa}%
```

```

1990 \ifnum\fc@min@weight<0 %
1991   \PackageError{fmtcount}{Out of range}%
1992   {This macro does not work with fractional numbers}%
1993 \fi
1994 \ifnum\fc@sign@case>0 %
1995   \PackageError{fmtcount}{Out of range}%
1996   {This macro does with negative or explicitly marked as positive numbers}%
1997 \fi

```

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

```

1998 \ifnum\fc@max@weight=0 %
1999   \ifnum\csname fc@digit@0\endcsname=1 %
2000     \count0=1 %
2001   \else
2002     \count0=0 %
2003   \fi
2004 \else
2005   \count0=0 %
2006 \fi
2007 \ifnum\count0=1 %
2008   \expandafter\@firstoftwo
2009 \else
2010   \expandafter\@secondoftwo
2011 \fi
2012
2013   {%
2014   \protected@edef\@tempa{\expandafter\fc@wcase\fc@first\@nil}%
2015   }%

```

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

```

2015   {%
2016   \def\@tempa##1{%
2017     \expandafter\edef\csname fc@frenchoptions@##1@plural\endcsname{%
2018       \ifcase\csname fc@frenchoptions@##1@plural\endcsname\space
2019       0% 0: always => always
2020       \or
2021       1% 1: never => never
2022       \or
2023       6% 2: multiple => multiple ng-last
2024       \or
2025       1% 3: multiple g-last => never
2026       \or
2027       5% 4: multiple l-last => multiple lng-last
2028       \or
2029       5% 5: multiple lng-last => multiple lng-last
2030       \or
2031       6% 6: multiple ng-last => multiple ng-last
2032     \fi

```

```

2033     }%
2034 }%
2035 \@tempa{vingt}%
2036 \@tempa{cent}%
2037 \@tempa{mil}%
2038 \@tempa{n-illion}%
2039 \@tempa{n-illiard}%

```

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

```

2040 \let\fc@wcase@save\fc@wcase
2041 \def\fc@wcase{\noexpand\fc@wcase}%
2042 \def\@nil{\noexpand\@nil}%

```

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

```

2043 \let\@tempa\@empty
2044 \fc@@nbrstrfrench@inner

```

Now restore \fc@wcase

```

2045 \let\fc@wcase\fc@wcase@save

```

Now we add the “ième” ending

```

2046 \expandafter\fc@get@last@word\expandafter{\@tempa}\@tempb\@tempc
2047 \expandafter\fc@get@last@letter\expandafter{\@tempc}\@tempd\@tempe
2048 \def\@tempf{e}%
2049 \ifx\@tempe\@tempf
2050 \protected@edef\@tempa{\@tempb\expandafter\fc@wcase\@tempd i\protect\'eme\@nil}%
2051 \else
2052 \def\@tempf{q}%
2053 \ifx\@tempe\@tempf
2054 \protected@edef\@tempa{\@tempb\expandafter\fc@wcase\@tempd qui\protect\'eme\@nil}%
2055 \else
2056 \def\@tempf{f}%
2057 \ifx\@tempe\@tempf
2058 \protected@edef\@tempa{\@tempb\expandafter\fc@wcase\@tempd vi\protect\'eme\@nil}%
2059 \else
2060 \protected@edef\@tempa{\@tempb\expandafter\fc@wcase\@tempc i\protect\'eme\@nil}%
2061 \fi
2062 \fi
2063 \fi
2064 }%

```

Apply \fc@gc case to the result.

```

2065 \fc@apply@gc case

```

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

```

2066 \def\@tempb##1{\def\@tempa{\def#2{##1}}}%
2067 \expandafter\@tempb\expandafter{\@tempa}%
2068 \expandafter
2069 }\@tempa
2070 }%
2071 \global\let\@@ordinalstringfrench\@@ordinalstringfrench

```

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

```
2072 \newcommand*\fc@frenchoptions@setdefaults{%
2073   \csname KV@fcfrench@all plural\endcsname{reformed}%
2074   \fc@gl@def\fc@frenchoptions@submillion@dos{-}%
2075   \fc@gl@let\fc@frenchoptions@supermillion@dos\space
2076   \fc@gl@let\fc@u@in@duo@empty% Could be ‘u’
2077   % \fc@gl@let\fc@poweroften\fc@pot@longscalefrench
2078   \fc@gl@let\fc@poweroften\fc@pot@recursivefrench
2079   \fc@gl@def\fc@longscale@nilliard@upto{0}% infinity
2080   \fc@gl@def\fc@frenchoptions@mil@plural@mark{le}%
2081 }%
2082 \global\let\fc@frenchoptions@setdefaults\fc@frenchoptions@setdefaults
2083 {%
2084   \let\fc@gl@def\gdef
2085   \def\fc@gl@let{\global\let}%
2086   \fc@frenchoptions@setdefaults
2087 }%
```

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

```
2088 \gdef\@ordinalstringMfrench{\csuse{\@ordinalstringMfrench\fmtcount@french}}%
2089 \gdef\@ordinalstringFfrench{\csuse{\@ordinalstringFfrench\fmtcount@french}}%
2090 \gdef\@OrdinalstringMfrench{\csuse{\@OrdinalstringMfrench\fmtcount@french}}%
2091 \gdef\@OrdinalstringFfrench{\csuse{\@OrdinalstringFfrench\fmtcount@french}}%
2092 \gdef\@numberstringMfrench{\csuse{\@numberstringMfrench\fmtcount@french}}%
2093 \gdef\@numberstringFfrench{\csuse{\@numberstringFfrench\fmtcount@french}}%
2094 \gdef\@NumberstringMfrench{\csuse{\@NumberstringMfrench\fmtcount@french}}%
2095 \gdef\@NumberstringFfrench{\csuse{\@NumberstringFfrench\fmtcount@french}}%
```

10.1.8 fc-frenchb.def

```
2096 \ProvidesFCLanguage{frenchb}[2013/08/17]%
2097 \FCloadlang{french}%
```

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

```
2098 \global\let\@ordinalMfrenchb=\@ordinalMfrench
2099 \global\let\@ordinalFfrenchb=\@ordinalFfrench
2100 \global\let\@ordinalNfrenchb=\@ordinalNfrench
2101 \global\let\@numberstringMfrenchb=\@numberstringMfrench
2102 \global\let\@numberstringFfrenchb=\@numberstringFfrench
2103 \global\let\@numberstringNfrenchb=\@numberstringNfrench
2104 \global\let\@NumberstringMfrenchb=\@NumberstringMfrench
2105 \global\let\@NumberstringFfrenchb=\@NumberstringFfrench
2106 \global\let\@NumberstringNfrenchb=\@NumberstringNfrench
2107 \global\let\@ordinalstringMfrenchb=\@ordinalstringMfrench
2108 \global\let\@ordinalstringFfrenchb=\@ordinalstringFfrench
2109 \global\let\@ordinalstringNfrenchb=\@ordinalstringNfrench
2110 \global\let\@OrdinalstringMfrenchb=\@OrdinalstringMfrench
2111 \global\let\@OrdinalstringFfrenchb=\@OrdinalstringFfrench
2112 \global\let\@OrdinalstringNfrenchb=\@OrdinalstringNfrench
```

10.1.9 fc-german.def

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

```
2113 \ProvidesFCLanguage{german}[2018/06/17]%
```

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

```
2114 \newcommand{\@ordinalMgerman}[2]{%
2115   \edef#2{\number#1\relax.}%
2116 }%
2117 \global\let\@ordinalMgerman\@ordinalMgerman
```

Feminine:

```
2118 \newcommand{\@ordinalFgerman}[2]{%
2119   \edef#2{\number#1\relax.}%
2120 }%
2121 \global\let\@ordinalFgerman\@ordinalFgerman
```

Neuter:

```
2122 \newcommand{\@ordinalNgerman}[2]{%
2123   \edef#2{\number#1\relax.}%
2124 }%
2125 \global\let\@ordinalNgerman\@ordinalNgerman
```

Convert a number to text. The easiest way to do this is to break it up into units, tens and teens.

Units (argument must be a number from 0 to 9, 1 on its own (eins) is dealt with separately):

```
2126 \newcommand*{\@unitstringgerman}[1]{%
2127   \ifcase#1%
2128     null%
2129     \or ein%
2130     \or zwei%
2131     \or drei%
2132     \or vier%
2133     \or fünf%
2134     \or sechs%
2135     \or sieben%
2136     \or acht%
2137     \or neun%
2138   \fi
2139 }%
2140 \global\let\@unitstringgerman\@unitstringgerman
```

Tens (argument must go from 1 to 10):

```
2141 \newcommand*{\@tenstringgerman}[1]{%
2142   \ifcase#1%
2143     \or zehn%
2144     \or zwanzig%
2145     \or dreißig%
2146     \or vierzig%
2147     \or fünfzig%
2148     \or sechzig%
```

```

2149 \or siebzig%
2150 \or achtzig%
2151 \or neunzig%
2152 \or einhundert%
2153 \fi
2154 }%
2155 \global\let\@@tenstringgerman\@@tenstringgerman

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

2156 \providecommand*\einhundert{\einhundert}%
2157 \providecommand*\eintausend{\eintausend}%
2158 \global\let\einhundert\einhundert
2159 \global\let\eintausend\eintausend

Teens:
2160 \newcommand*\@@teenstringgerman[1]{%
2161 \ifcase#1%
2162 zehn%
2163 \or elf%
2164 \or zwölf%
2165 \or dreizehn%
2166 \or vierzehn%
2167 \or fünfzehn%
2168 \or sechzehn%
2169 \or siebzehn%
2170 \or achtzehn%
2171 \or neunzehn%
2172 \fi
2173 }%
2174 \global\let\@@teenstringgerman\@@teenstringgerman

The results are stored in the second argument, but doesn't display anything.
2175 \newcommand*\@numberstringMgerman}[2]{%
2176 \let\@unitstring=\@unitstringgerman
2177 \let\@teenstring=\@teenstringgerman
2178 \let\@tenstring=\@tenstringgerman
2179 \@numberstringgerman{#1}{#2}%
2180 }%
2181 \global\let\@numberstringMgerman\@numberstringMgerman

Feminine and neuter forms:
2182 \global\let\@numberstringFgerman=\@numberstringMgerman
2183 \global\let\@numberstringNgerman=\@numberstringMgerman

As above, but initial letters in upper case:
2184 \newcommand*\@NumberstringMgerman}[2]{%
2185 \@numberstringMgerman{#1}{\@num@str}%
2186 \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
2187 }%
2188 \global\let\@NumberstringMgerman\@NumberstringMgerman

```

Feminine and neuter form:

```
2189 \global\let\@NumberstringFgerman=\@NumberstringMgerman
2190 \global\let\@NumberstringNgerman=\@NumberstringMgerman
```

As above, but for ordinals.

```
2191 \newcommand*\@ordinalstringMgerman}[2]{%
2192   \let\@unitthstring=\@unitthstringMgerman
2193   \let\@teenthstring=\@teenthstringMgerman
2194   \let\@tenthstring=\@tenthstringMgerman
2195   \let\@unitstring=\@unitstringgerman
2196   \let\@teenstring=\@teenstringgerman
2197   \let\@tenstring=\@tenstringgerman
2198   \def\@thousandth{tausendster}%
2199   \def\@hundredth{hundertster}%
2200   \@ordinalstringgerman{#1}{#2}%
2201 }%
2202 \global\let\@ordinalstringMgerman\@ordinalstringMgerman
```

Feminine form:

```
2203 \newcommand*\@ordinalstringFgerman}[2]{%
2204   \let\@unitthstring=\@unitthstringFgerman
2205   \let\@teenthstring=\@teenthstringFgerman
2206   \let\@tenthstring=\@tenthstringFgerman
2207   \let\@unitstring=\@unitstringgerman
2208   \let\@teenstring=\@teenstringgerman
2209   \let\@tenstring=\@tenstringgerman
2210   \def\@thousandth{tausendste}%
2211   \def\@hundredth{hundertste}%
2212   \@ordinalstringgerman{#1}{#2}%
2213 }%
2214 \global\let\@ordinalstringFgerman\@ordinalstringFgerman
```

Neuter form:

```
2215 \newcommand*\@ordinalstringNgerman}[2]{%
2216   \let\@unitthstring=\@unitthstringNgerman
2217   \let\@teenthstring=\@teenthstringNgerman
2218   \let\@tenthstring=\@tenthstringNgerman
2219   \let\@unitstring=\@unitstringgerman
2220   \let\@teenstring=\@teenstringgerman
2221   \let\@tenstring=\@tenstringgerman
2222   \def\@thousandth{tausendstes}%
2223   \def\@hundredth{hunderstes}%
2224   \@ordinalstringgerman{#1}{#2}%
2225 }%
2226 \global\let\@ordinalstringNgerman\@ordinalstringNgerman
```

As above, but with initial letters in upper case.

```
2227 \newcommand*\@OrdinalstringMgerman}[2]{%
2228   \@ordinalstringMgerman{#1}{\@num@str}%
2229   \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
2230 }%
```



```
2231 \global\let\@OrdinalstringMgerman\@OrdinalstringMgerman
```

Feminine form:

```
2232 \newcommand*\@OrdinalstringFgerman}[2]{%
2233 \@OrdinalstringFgerman{#1}{\@num@str}%
2234 \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
2235 }%
2236 \global\let\@OrdinalstringFgerman\@OrdinalstringFgerman
```

Neuter form:

```
2237 \newcommand*\@OrdinalstringNgerman}[2]{%
2238 \@OrdinalstringNgerman{#1}{\@num@str}%
2239 \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
2240 }%
2241 \global\let\@OrdinalstringNgerman\@OrdinalstringNgerman
```

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

```
2242 \newcommand*\@unitthstringMgerman[1]{%
2243 \ifcase#1%
2244 nullter%
2245 \or erster%
2246 \or zweiter%
2247 \or dritter%
2248 \or vierter%
2249 \or fünfter%
2250 \or sechster%
2251 \or siebter%
2252 \or achter%
2253 \or neunter%
2254 \fi
2255 }%
2256 \global\let\@unitthstringMgerman\@unitthstringMgerman
```

Tens:

```
2257 \newcommand*\@tenthstringMgerman[1]{%
2258 \ifcase#1%
2259 \or zehnter%
2260 \or zwanzigster%
2261 \or dreißigster%
2262 \or vierzigster%
2263 \or fünfzigster%
2264 \or sechzigster%
2265 \or siebzigster%
2266 \or achtzigster%
2267 \or neunzigster%
2268 \fi
2269 }%
2270 \global\let\@tenthstringMgerman\@tenthstringMgerman
```

Teens:

```

2271 \newcommand*\@@teenthstringMgerman[1]{%
2272   \ifcase#1%
2273     zehnter%
2274     \or elfter%
2275     \or zwölfter%
2276     \or dreizehnter%
2277     \or vierzehnter%
2278     \or fünfzehnter%
2279     \or sechzehnter%
2280     \or siebzehnter%
2281     \or achtzehnter%
2282     \or neunzehnter%
2283   \fi
2284 }%
2285 \global\let\@@teenthstringMgerman\@@teenthstringMgerman

Units (feminine):
2286 \newcommand*\@@unitthstringFgerman[1]{%
2287   \ifcase#1%
2288     nullte%
2289     \or erste%
2290     \or zweite%
2291     \or dritte%
2292     \or vierte%
2293     \or fünfte%
2294     \or sechste%
2295     \or siebte%
2296     \or achte%
2297     \or neunte%
2298   \fi
2299 }%
2300 \global\let\@@unitthstringFgerman\@@unitthstringFgerman

Tens (feminine):
2301 \newcommand*\@@tenthstringFgerman[1]{%
2302   \ifcase#1%
2303     \or zehnte%
2304     \or zwanzigste%
2305     \or dreißigste%
2306     \or vierzigste%
2307     \or fünfzigste%
2308     \or sechzigste%
2309     \or siebzigste%
2310     \or achtzigste%
2311     \or neunzigste%
2312   \fi
2313 }%
2314 \global\let\@@tenthstringFgerman\@@tenthstringFgerman

Teens (feminine)
2315 \newcommand*\@@teenthstringFgerman[1]{%

```

```

2316 \ifcase#1%
2317     zehnte%
2318     \or elfte%
2319     \or zwölfte%
2320     \or dreizehnte%
2321     \or vierzehnte%
2322     \or fünfzehnte%
2323     \or sechzehnte%
2324     \or siebzehnte%
2325     \or achtzehnte%
2326     \or neunzehnte%
2327 \fi
2328 }%
2329 \global\let\@@teenthstringFgerman\@@teenthstringFgerman

```

Units (neuter):

```

2330 \newcommand*\@@unitthstringNgerman[1]{%
2331     \ifcase#1%
2332         nulltes%
2333         \or erstes%
2334         \or zweites%
2335         \or drittes%
2336         \or viertes%
2337         \or fünftes%
2338         \or sechstes%
2339         \or siebttes%
2340         \or achttes%
2341         \or neuntes%
2342 \fi
2343 }%
2344 \global\let\@@unitthstringNgerman\@@unitthstringNgerman

```

Tens (neuter):

```

2345 \newcommand*\@@tenthstringNgerman[1]{%
2346     \ifcase#1%
2347         \or zehntes%
2348         \or zwanzigstes%
2349         \or dreißigstes%
2350         \or vierzigstes%
2351         \or fünfzigstes%
2352         \or sechzigstes%
2353         \or siebzigstes%
2354         \or achtzigstes%
2355         \or neunzigstes%
2356 \fi
2357 }%
2358 \global\let\@@tenthstringNgerman\@@tenthstringNgerman

```

Teens (neuter)

```

2359 \newcommand*\@@teenthstringNgerman[1]{%
2360     \ifcase#1%

```

```

2361     zehntes%
2362     \or elftes%
2363     \or zwölftes%
2364     \or dreizehntes%
2365     \or vierzehntes%
2366     \or fünfzehntes%
2367     \or sechzehntes%
2368     \or siebzehntes%
2369     \or achtzehntes%
2370     \or neunzehntes%
2371 \fi
2372 }%
2373 \global\let\@@teenthstringNgerman\@@teenthstringNgerman

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

2374 \newcommand*\@@numberunderhundredgerman[2]{%
2375 \ifnum#1<10\relax
2376   \ifnum#1>0\relax
2377     \eappto#2{\@unitstring{#1}}%
2378   \fi
2379 \else
2380   \@tmpstrctr=#1\relax
2381   \@FCmodulo{\@tmpstrctr}{10}%
2382   \ifnum#1<20\relax
2383     \eappto#2{\@teenstring{\@tmpstrctr}}%
2384   \else
2385     \ifnum\@tmpstrctr=0\relax
2386     \else
2387       \eappto#2{\@unitstring{\@tmpstrctr}und}%
2388     \fi
2389     \@tmpstrctr=#1\relax
2390     \divide\@tmpstrctr by 10\relax
2391     \eappto#2{\@tenstring{\@tmpstrctr}}%
2392   \fi
2393 \fi
2394 }%
2395 \global\let\@@numberunderhundredgerman\@@numberunderhundredgerman

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

2396 \newcommand*\@@numberstringgerman[2]{%
2397 \ifnum#1>99999\relax
2398   \PackageError{fmtcount}{Out of range}%
2399   {This macro only works for values less than 100000}%
2400 \else
2401   \ifnum#1<0\relax
2402     \PackageError{fmtcount}{Negative numbers not permitted}%
2403     {This macro does not work for negative numbers, however
2404     you can try typing "minus" first, and then pass the modulus of

```

```

2405     this number}%
2406   \fi
2407 \fi
2408 \def#2{}%
2409 \@strctr=#1\relax \divide\@strctr by 1000\relax
2410 \ifnum\@strctr>1\relax
    #1 is  $\geq 2000$ , \@strctr now contains the number of thousands
2411   \@@numberunderhundredgerman{\@strctr}{#2}%
2412   \appto#2{tausend}%
2413 \else
    #1 lies in range [1000,1999]
2414   \ifnum\@strctr=1\relax
2415     \eappto#2{\eintausend}%
2416   \fi
2417 \fi
2418 \@strctr=#1\relax
2419 \@FCmodulo{\@strctr}{1000}%
2420 \divide\@strctr by 100\relax
2421 \ifnum\@strctr>1\relax
    now dealing with number in range [200,999]
2422   \eappto#2{\@unitstring{\@strctr}hundert}%
2423 \else
2424   \ifnum\@strctr=1\relax
    dealing with number in range [100,199]
2425     \ifnum#1>1000\relax
    if original number > 1000, use einhundert
2426       \appto#2{einhundert}%
2427     \else
    otherwise use \einhundert
2428       \eappto#2{\einhundert}%
2429     \fi
2430   \fi
2431 \fi
2432 \@strctr=#1\relax
2433 \@FCmodulo{\@strctr}{100}%
2434 \ifnum#1=0\relax
2435   \def#2{null}%
2436 \else
2437   \ifnum\@strctr=1\relax
2438     \appto#2{eins}%
2439   \else
2440     \@@numberunderhundredgerman{\@strctr}{#2}%
2441   \fi
2442 \fi
2443 }%
2444 \global\let\@@numberstringgerman\@@numberstringgerman

```

As above, but for ordinals

```
2445 \newcommand*\@@numberunderhundredthgerman[2]{%
2446 \ifnum#1<10\relax
2447 \eappto#2{\@unitthstring{#1}}%
2448 \else
2449 \@tmpstrctr=#1\relax
2450 \@FCmodulo{\@tmpstrctr}{10}%
2451 \ifnum#1<20\relax
2452 \eappto#2{\@teenthstring{\@tmpstrctr}}%
2453 \else
2454 \ifnum\@tmpstrctr=0\relax
2455 \else
2456 \eappto#2{\@unitstring{\@tmpstrctr}und}%
2457 \fi
2458 \@tmpstrctr=#1\relax
2459 \divide\@tmpstrctr by 10\relax
2460 \eappto#2{\@tentstring{\@tmpstrctr}}%
2461 \fi
2462 \fi
2463 }%
2464 \global\let\@@numberunderhundredthgerman\@@numberunderhundredthgerman

2465 \newcommand*\@@ordinalstringgerman[2]{%
2466 \@orgargctr=#1\relax
2467 \ifnum\@orgargctr>99999\relax
2468 \PackageError{fmtcount}{Out of range}%
2469 {This macro only works for values less than 100000}%
2470 \else
2471 \ifnum\@orgargctr<0\relax
2472 \PackageError{fmtcount}{Negative numbers not permitted}%
2473 {This macro does not work for negative numbers, however
2474 you can try typing "minus" first, and then pass the modulus of
2475 this number}%
2476 \fi
2477 \fi
2478 \def#2{}%
2479 \@strctr=\@orgargctr\divide\@strctr by 1000\relax
2480 \ifnum\@strctr>1\relax

#1 is  $\geq 2000$ , \@strctr now contains the number of thousands
2481 \@@numberunderhundredthgerman{\@strctr}{#2}%

is that it, or is there more?
2482 \@tmpstrctr=\@orgargctr\@FCmodulo{\@tmpstrctr}{1000}%
2483 \ifnum\@tmpstrctr=0\relax
2484 \eappto#2{\@thousandth}%
2485 \else
2486 \appto#2{tausend}%
2487 \fi
2488 \else
```

```

#1 lies in range [1000,1999]
2489 \ifnum\@strctr=1\relax
2490 \ifnum\@orgargctr=1000\relax
2491 \eappto#2{\@thousandth}%
2492 \else
2493 \eappto#2{\eintausend}%
2494 \fi
2495 \fi
2496 \fi
2497 \@strctr=\@orgargctr
2498 \@FCmodulo{\@strctr}{1000}%
2499 \divide\@strctr by 100\relax
2500 \ifnum\@strctr>1\relax
    now dealing with number in range [200,999] is that it, or is there more?
2501 \@tmpstrctr=\@orgargctr \@FCmodulo{\@tmpstrctr}{100}%
2502 \ifnum\@tmpstrctr=0\relax
2503 \ifnum\@strctr=1\relax
2504 \eappto#2{\@hundredth}%
2505 \else
2506 \eappto#2{\@unitstring{\@strctr}\@hundredth}%
2507 \fi
2508 \else
2509 \eappto#2{\@unitstring{\@strctr}hundert}%
2510 \fi
2511 \else
2512 \ifnum\@strctr=1\relax
    dealing with number in range [100,199] is that it, or is there more?
2513 \@tmpstrctr=\@orgargctr \@FCmodulo{\@tmpstrctr}{100}%
2514 \ifnum\@tmpstrctr=0\relax
2515 \eappto#2{\@hundredth}%
2516 \else
2517 \ifnum\@orgargctr>1000\relax
2518 \appto#2{einhundert}%
2519 \else
2520 \eappto#2{\einhundert}%
2521 \fi
2522 \fi
2523 \fi
2524 \fi
2525 \@strctr=\@orgargctr
2526 \@FCmodulo{\@strctr}{100}%
2527 \ifthenelse{\@strctr=0 \and \@orgargctr>0 }{ }{%
2528 \@@numberunderhundredthgerman{\@strctr}{#2}%
2529 }%
2530 }%
2531 \global\let\@@ordinalstringgerman\@@ordinalstringgerman
    Load fc-germanb.def if not already loaded
2532 \FCloadlang{germanb}%

```

10.1.10 fc-germanb.def

```
2533 \ProvidesFCLanguage{germanb}[2013/08/17]%
```

Load fc-german.def if not already loaded

```
2534 \FCloadlang{german}%
```

Set |germanb| to be equivalent to |german|.

```
2535 \global\let\@ordinalMgermanb=\@ordinalMgerman
```

```
2536 \global\let\@ordinalFgermanb=\@ordinalFgerman
```

```
2537 \global\let\@ordinalNgermanb=\@ordinalNgerman
```

```
2538 \global\let\@numberstringMgermanb=\@numberstringMgerman
```

```
2539 \global\let\@numberstringFgermanb=\@numberstringFgerman
```

```
2540 \global\let\@numberstringNgermanb=\@numberstringNgerman
```

```
2541 \global\let\@NumberstringMgermanb=\@NumberstringMgerman
```

```
2542 \global\let\@NumberstringFgermanb=\@NumberstringFgerman
```

```
2543 \global\let\@NumberstringNgermanb=\@NumberstringNgerman
```

```
2544 \global\let\@ordinalstringMgermanb=\@ordinalstringMgerman
```

```
2545 \global\let\@ordinalstringFgermanb=\@ordinalstringFgerman
```

```
2546 \global\let\@ordinalstringNgermanb=\@ordinalstringNgerman
```

```
2547 \global\let\@OrdinalstringMgermanb=\@OrdinalstringMgerman
```

```
2548 \global\let\@OrdinalstringFgermanb=\@OrdinalstringFgerman
```

```
2549 \global\let\@OrdinalstringNgermanb=\@OrdinalstringNgerman
```

10.1.11 fc-italian

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

```
2550 \ProvidesFCLanguage{italian}[2013/08/17]
```

```
2551
```

```
2552 \RequirePackage{itnumpar}
```

```
2553
```

```
2554 \newcommand{\@numberstringMitalian}[2]{%
```

```
2555   \begingroup
```

```
2556     \def\np@oa{o}%
```

```
2557     \count@=#1
```

```
2558     \edef\@tempa{\def\noexpand#2{\@numeroinparole{\count@}}}%
```

```
2559     \expandafter
```

```
2560   \endgroup\@tempa
```

```
2561 }
```

```
2562 \global\let\@numberstringMitalian\@numberstringMitalian
```

```
2563
```

```
2564 \newcommand{\@numberstringFitalian}[2]{%
```

```
2565   \begingroup
```

```
2566     \def\np@oa{a}%
```

```
2567     \count@=#1
```

```
2568     \edef\@tempa{\def\noexpand#2{\@numeroinparole{\count@}}}%
```

```
2569     \expandafter
```

```
2570   \endgroup\@tempa
```

```
2571 }
```

```
2572
```

```
2573 \global\let\@numberstringFitalian\@numberstringFitalian
```



```

2574
2575 \newcommand{\@NumberstringMitalian}[2]{%
2576   \begingroup
2577     \def\np@oa{o}%
2578     \count@=#1
2579     \edef\@tempa{\def\noexpand#2{\@Numeroinparole{\count@}}}%
2580     \expandafter
2581   \endgroup\@tempa
2582 }
2583 \global\let\@NumberstringMitalian\@NumberstringMitalian
2584
2585 \newcommand{\@NumberstringFitalian}[2]{%
2586   \begingroup
2587     \def\np@oa{a}%
2588     \count@=#1
2589     \edef\@tempa{\def\noexpand#2{\@Numeroinparole{\count@}}}%
2590     \expandafter
2591   \endgroup\@tempa
2592 }
2593 \global\let\@NumberstringFitalian\@NumberstringFitalian
2594
2595 \newcommand{\@OrdinalstringMitalian}[2]{%
2596   \begingroup
2597     \count@=#1
2598     \edef\@tempa{\def\noexpand#2{\@Ordinalef{\count@}}}%
2599     \expandafter
2600   \endgroup\@tempa
2601 }
2602 \global\let\@OrdinalstringMitalian\@OrdinalstringMitalian
2603
2604 \newcommand{\@OrdinalstringFitalian}[2]{%
2605   \begingroup
2606     \count@=#1
2607     \edef\@tempa{\def\noexpand#2{\@Ordinalef{\count@}}}%
2608     \expandafter
2609   \endgroup\@tempa
2610 }
2611 \global\let\@OrdinalstringFitalian\@OrdinalstringFitalian
2612
2613 \newcommand{\@OrdinalstringMitalian}[2]{%
2614   \begingroup
2615     \count@=#1
2616     \edef\@tempa{\def\noexpand#2{\@Ordinalef{\count@}}}%
2617     \expandafter
2618   \endgroup\@tempa
2619 }
2620 \global\let\@OrdinalstringMitalian\@OrdinalstringMitalian
2621
2622 \newcommand{\@OrdinalstringFitalian}[2]{%

```

```

2623 \begingroup
2624   \count@=#1
2625   \edef\@tempa{\def\noexpand#2{\@Ordinalef{\count@}}}%
2626   \expandafter
2627   \endgroup\@tempa
2628 }
2629 \global\let\@OrdinalstringFitalian\@OrdinalstringFitalian
2630
2631 \newcommand{\@ordinalMitalian}[2]{%
2632   \edef#2{#1\relax\noexpand\fmtord{o}}%
2633
2634 \global\let\@ordinalMitalian\@ordinalMitalian
2635
2636 \newcommand{\@ordinalFitalian}[2]{%
2637   \edef#2{#1\relax\noexpand\fmtord{a}}%
2638 \global\let\@ordinalFitalian\@ordinalFitalian

```

10.1.12 fc-ngerman.def

```

2639 \ProvidesFCLanguage{ngerman}[2012/06/18]%
2640 \FCloadlang{german}%
2641 \FCloadlang{ngermanb}%

```

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

```

2642 \global\let\@ordinalMngerman=\@ordinalMgerman
2643 \global\let\@ordinalFngerman=\@ordinalFgerman
2644 \global\let\@ordinalNngerman=\@ordinalNgerman
2645 \global\let\@numberstringMngerman=\@numberstringMgerman
2646 \global\let\@numberstringFngerman=\@numberstringFgerman
2647 \global\let\@numberstringNngerman=\@numberstringNgerman
2648 \global\let\@NumberstringMngerman=\@NumberstringMgerman
2649 \global\let\@NumberstringFngerman=\@NumberstringFgerman
2650 \global\let\@NumberstringNngerman=\@NumberstringNgerman
2651 \global\let\@ordinalstringMngerman=\@ordinalstringMgerman
2652 \global\let\@ordinalstringFngerman=\@ordinalstringFgerman
2653 \global\let\@ordinalstringNngerman=\@ordinalstringNgerman
2654 \global\let\@OrdinalstringMngerman=\@OrdinalstringMgerman
2655 \global\let\@OrdinalstringFngerman=\@OrdinalstringFgerman
2656 \global\let\@OrdinalstringNngerman=\@OrdinalstringNgerman

```

10.1.13 fc-ngermanb.def

```

2657 \ProvidesFCLanguage{ngermanb}[2013/08/17]%
2658 \FCloadlang{german}%

```

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

```

2659 \global\let\@ordinalMngermanb=\@ordinalMgerman
2660 \global\let\@ordinalFngermanb=\@ordinalFgerman
2661 \global\let\@ordinalNngermanb=\@ordinalNgerman

```

```

2662 \global\let\@numberstringMngermanb=\@numberstringMgerman
2663 \global\let\@numberstringFngermanb=\@numberstringFgerman
2664 \global\let\@numberstringNngermanb=\@numberstringNgerman
2665 \global\let\@NumberstringMngermanb=\@NumberstringMgerman
2666 \global\let\@NumberstringFngermanb=\@NumberstringFgerman
2667 \global\let\@NumberstringNngermanb=\@NumberstringNgerman
2668 \global\let\@ordinalstringMngermanb=\@ordinalstringMgerman
2669 \global\let\@ordinalstringFngermanb=\@ordinalstringFgerman
2670 \global\let\@ordinalstringNngermanb=\@ordinalstringNgerman
2671 \global\let\@OrdinalstringMngermanb=\@OrdinalstringMgerman
2672 \global\let\@OrdinalstringFngermanb=\@OrdinalstringFgerman
2673 \global\let\@OrdinalstringNngermanb=\@OrdinalstringNgerman

```

Load fc-ngerman.def if not already loaded

```
2674 \FCloadlang{ngerman}%
```

10.1.14 fc-portuges.def

Portuguese definitions

```
2675 \ProvidesFCLanguage{portuges}[2017/12/26]%
```

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:

```

2676 \newcommand*\@ordinalMportuges[2]{%
2677   \ifnum#1=0\relax
2678     \edef#2{\number#1}%
2679   \else
2680     \edef#2{\number#1\relax\noexpand\fmtord{o}}%
2681   \fi
2682 }%
2683 \global\let\@ordinalMportuges\@ordinalMportuges

```

Feminine:

```

2684 \newcommand*\@ordinalFportuges[2]{%
2685   \ifnum#1=0\relax
2686     \edef#2{\number#1}%
2687   \else
2688     \edef#2{\number#1\relax\noexpand\fmtord{a}}%
2689   \fi
2690 }%
2691 \global\let\@ordinalFportuges\@ordinalFportuges

```

Make neuter same as masculine:

```
2692 \global\let\@ordinalNportuges\@ordinalMportuges
```

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

```

2693 \newcommand*\@@unitstringportuges[1]{%
2694   \ifcase#1\relax
2695     zero%
2696   \or um%
2697   \or dois%

```

```

2698 \or tr\^es%
2699 \or quatro%
2700 \or cinco%
2701 \or seis%
2702 \or sete%
2703 \or oito%
2704 \or nove%
2705 \fi
2706 }%
2707 \global\let\@@unitstringportuges\@@unitstringportuges
2708 % \end{macrocode}
2709 % As above, but for feminine:
2710 % \begin{macrocode}
2711 \newcommand*\@@unitstringFportuges[1]{%
2712 \ifcase#1\relax
2713 zero%
2714 \or uma%
2715 \or duas%
2716 \or tr\^es%
2717 \or quatro%
2718 \or cinco%
2719 \or seis%
2720 \or sete%
2721 \or oito%
2722 \or nove%
2723 \fi
2724 }%
2725 \global\let\@@unitstringFportuges\@@unitstringFportuges
Tens (argument must be a number from 0 to 10):
2726 \newcommand*\@@tenstringportuges[1]{%
2727 \ifcase#1\relax
2728 \or dez%
2729 \or vinte%
2730 \or trinta%
2731 \or quarenta%
2732 \or cinq\"uenta%
2733 \or sessenta%
2734 \or setenta%
2735 \or oitenta%
2736 \or noventa%
2737 \or cem%
2738 \fi
2739 }%
2740 \global\let\@@tenstringportuges\@@tenstringportuges
Teens (argument must be a number from 0 to 9):
2741 \newcommand*\@@teenstringportuges[1]{%
2742 \ifcase#1\relax
2743 dez%

```

2744 \or onze%
 2745 \or doze%
 2746 \or treze%
 2747 \or catorze%
 2748 \or quinze%
 2749 \or dezasseis%
 2750 \or dezassete%
 2751 \or dezoito%
 2752 \or dezanove%
 2753 \fi
 2754 }%
 2755 \global\let\@@teenstringportuges\@@teenstringportuges

Hundreds:

2756 \newcommand*\@@hundredstringportuges[1]{%
 2757 \ifcase#1\relax
 2758 \or cento%
 2759 \or duzentos%
 2760 \or trezentos%
 2761 \or quatrocentos%
 2762 \or quinhentos%
 2763 \or seiscentos%
 2764 \or setecentos%
 2765 \or oitocentos%
 2766 \or novecentos%
 2767 \fi
 2768 }%
 2769 \global\let\@@hundredstringportuges\@@hundredstringportuges

Hundreds (feminine):

2770 \newcommand*\@@hundredstringFportuges[1]{%
 2771 \ifcase#1\relax
 2772 \or cento%
 2773 \or duzentas%
 2774 \or trezentas%
 2775 \or quatrocentas%
 2776 \or quinhentas%
 2777 \or seiscentas%
 2778 \or setecentas%
 2779 \or oitocentas%
 2780 \or novecentas%
 2781 \fi
 2782 }%
 2783 \global\let\@@hundredstringFportuges\@@hundredstringFportuges

Units (initial letter in upper case):

2784 \newcommand*\@@Unitstringportuges[1]{%
 2785 \ifcase#1\relax
 2786 Zero%
 2787 \or Um%
 2788 \or Dois%

```

2789 \or Tr\~es%
2790 \or Quatro%
2791 \or Cinco%
2792 \or Seis%
2793 \or Sete%
2794 \or Oito%
2795 \or Nove%
2796 \fi
2797 }%
2798 \global\let\@@Unitstringportuges\@@Unitstringportuges

```

As above, but feminine:

```

2799 \newcommand*\@@UnitstringFportuges[1]{%
2800 \ifcase#1\relax
2801 Zera%
2802 \or Uma%
2803 \or Duas%
2804 \or Tr\~es%
2805 \or Quatro%
2806 \or Cinco%
2807 \or Seis%
2808 \or Sete%
2809 \or Oito%
2810 \or Nove%
2811 \fi
2812 }%
2813 \global\let\@@UnitstringFportuges\@@UnitstringFportuges

```

Tens (with initial letter in upper case):

```

2814 \newcommand*\@@Tenstringportuges[1]{%
2815 \ifcase#1\relax
2816 \or Dez%
2817 \or Vinte%
2818 \or Trinta%
2819 \or Quarenta%
2820 \or Cinq\"uenta%
2821 \or Sessenta%
2822 \or Setenta%
2823 \or Oitenta%
2824 \or Noventa%
2825 \or Cem%
2826 \fi
2827 }%
2828 \global\let\@@Tenstringportuges\@@Tenstringportuges

```

Teens (with initial letter in upper case):

```

2829 \newcommand*\@@Teenstringportuges[1]{%
2830 \ifcase#1\relax
2831 Dez%
2832 \or Onze%
2833 \or Doze%

```

```

2834 \or Treze%
2835 \or Catorze%
2836 \or Quinze%
2837 \or Dezasseis%
2838 \or Dezassete%
2839 \or Dezoito%
2840 \or Dezanove%
2841 \fi
2842 }%
2843 \global\let\@@Teenstringportuges\@@Teenstringportuges

Hundreds (with initial letter in upper case):
2844 \newcommand*\@@Hundredstringportuges[1]{%
2845 \ifcase#1\relax
2846 \or Cento%
2847 \or Duzentos%
2848 \or Trezentos%
2849 \or Quatrocentos%
2850 \or Quinhentos%
2851 \or Seiscentos%
2852 \or Setecentos%
2853 \or Oitocentos%
2854 \or Novecentos%
2855 \fi
2856 }%
2857 \global\let\@@Hundredstringportuges\@@Hundredstringportuges

As above, but feminine:
2858 \newcommand*\@@HundredstringFportuges[1]{%
2859 \ifcase#1\relax
2860 \or Cento%
2861 \or Duzentas%
2862 \or Trezentas%
2863 \or Quatrocentas%
2864 \or Quinhentas%
2865 \or Seiscentas%
2866 \or Setecentas%
2867 \or Oitocentas%
2868 \or Novecentas%
2869 \fi
2870 }%
2871 \global\let\@@HundredstringFportuges\@@HundredstringFportuges

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

2872 \newcommand*\@@numberstringMportuges}[2]{%
2873 \let\@unitstring=\@@unitstringportuges
2874 \let\@teenstring=\@@teenstringportuges
2875 \let\@tenstring=\@@tenstringportuges

```

```

2876 \let\@hundredstring=\@hundredstringportuges
2877 \def\@hundred{cem}\def\@thousand{mil}%
2878 \def\@andname{e}%
2879 \@numberstringportuges{#1}{#2}%
2880 }%
2881 \global\let\@numberstringMportuges\@numberstringMportuges

  As above, but feminine form:
2882 \newcommand*\@numberstringFportuges}[2]{%
2883 \let\@unitstring=\@unitstringFportuges
2884 \let\@teenstring=\@teenstringportuges
2885 \let\@tenstring=\@tenstringportuges
2886 \let\@hundredstring=\@hundredstringFportuges
2887 \def\@hundred{cem}\def\@thousand{mil}%
2888 \def\@andname{e}%
2889 \@numberstringportuges{#1}{#2}%
2890 }%
2891 \global\let\@numberstringFportuges\@numberstringFportuges

  Make neuter same as masculine:
2892 \global\let\@numberstringNportuges\@numberstringMportuges

  As above, but initial letters in upper case:
2893 \newcommand*\@NumberstringMportuges}[2]{%
2894 \let\@unitstring=\@Unitstringportuges
2895 \let\@teenstring=\@Teenstringportuges
2896 \let\@tenstring=\@Tenstringportuges
2897 \let\@hundredstring=\@Hundredstringportuges
2898 \def\@hundred{Cem}\def\@thousand{Mil}%
2899 \def\@andname{e}%
2900 \@numberstringportuges{#1}{#2}%
2901 }%
2902 \global\let\@NumberstringMportuges\@NumberstringMportuges

  As above, but feminine form:
2903 \newcommand*\@NumberstringFportuges}[2]{%
2904 \let\@unitstring=\@UnitstringFportuges
2905 \let\@teenstring=\@Teenstringportuges
2906 \let\@tenstring=\@Tenstringportuges
2907 \let\@hundredstring=\@HundredstringFportuges
2908 \def\@hundred{Cem}\def\@thousand{Mil}%
2909 \def\@andname{e}%
2910 \@numberstringportuges{#1}{#2}%
2911 }%
2912 \global\let\@NumberstringFportuges\@NumberstringFportuges

  Make neuter same as masculine:
2913 \global\let\@NumberstringNportuges\@NumberstringMportuges

  As above, but for ordinals.
2914 \newcommand*\@ordinalstringMportuges}[2]{%
2915 \let\@unitthstring=\@unitthstringportuges

```



```

2916 \let\@unitstring=\@unitstringportuges
2917 \let\@teenthstring=\@teenthstringportuges
2918 \let\@tenthstring=\@tenthstringportuges
2919 \let\@hundredthstring=\@hundredthstringportuges
2920 \def\@thousandth{mil\'esimo}%
2921 \@ordinalstringportuges{#1}{#2}%
2922 }%
2923 \global\let\@ordinalstringMportuges\@ordinalstringMportuges

```

Feminine form:

```

2924 \newcommand*\@ordinalstringFportuges}[2]{%
2925 \let\@unitthstring=\@unitthstringFportuges
2926 \let\@unitstring=\@unitstringFportuges
2927 \let\@teenthstring=\@teenthstringportuges
2928 \let\@tenthstring=\@tenthstringFportuges
2929 \let\@hundredthstring=\@hundredthstringFportuges
2930 \def\@thousandth{mil\'esima}%
2931 \@ordinalstringportuges{#1}{#2}%
2932 }%
2933 \global\let\@ordinalstringFportuges\@ordinalstringFportuges

```

Make neuter same as masculine:

```

2934 \global\let\@ordinalstringNportuges\@ordinalstringMportuges

```

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

```

2935 \newcommand*\@OrdinalstringMportuges}[2]{%
2936 \let\@unitthstring=\@Unitthstringportuges
2937 \let\@unitstring=\@Unitstringportuges
2938 \let\@teenthstring=\@teenthstringportuges
2939 \let\@tenthstring=\@Tenthstringportuges
2940 \let\@hundredthstring=\@Hundredthstringportuges
2941 \def\@thousandth{Mil\'esimo}%
2942 \@ordinalstringportuges{#1}{#2}%
2943 }%
2944 \global\let\@OrdinalstringMportuges\@OrdinalstringMportuges

```

Feminine form:

```

2945 \newcommand*\@OrdinalstringFportuges}[2]{%
2946 \let\@unitthstring=\@UnitthstringFportuges
2947 \let\@unitstring=\@UnitstringFportuges
2948 \let\@teenthstring=\@teenthstringportuges
2949 \let\@tenthstring=\@TenthstringFportuges
2950 \let\@hundredthstring=\@HundredthstringFportuges
2951 \def\@thousandth{Mil\'esima}%
2952 \@ordinalstringportuges{#1}{#2}%
2953 }%
2954 \global\let\@OrdinalstringFportuges\@OrdinalstringFportuges

```

Make neuter same as masculine:

```

2955 \global\let\@OrdinalstringNportuges\@OrdinalstringMportuges

```

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

```

2956 \newcommand*\@@unitthstringportuges[1]{%
2957   \ifcase#1\relax
2958     zero%
2959     \or primeiro%
2960     \or segundo%
2961     \or terceiro%
2962     \or quarto%
2963     \or quinto%
2964     \or sexto%
2965     \or s\'etimo%
2966     \or oitavo%
2967     \or nono%
2968   \fi
2969 }%
2970 \global\let\@@unitthstringportuges\@@unitthstringportuges

```

Tens:

```

2971 \newcommand*\@@tenthstringportuges[1]{%
2972   \ifcase#1\relax
2973     \or d\'ecimo%
2974     \or vig\'esimo%
2975     \or trig\'esimo%
2976     \or quadrag\'esimo%
2977     \or q\'uinquag\'esimo%
2978     \or sexag\'esimo%
2979     \or setuag\'esimo%
2980     \or octog\'esimo%
2981     \or nonag\'esimo%
2982   \fi
2983 }%
2984 \global\let\@@tenthstringportuges\@@tenthstringportuges

```

Teens:

```

2985 \newcommand*\@@teenthstringportuges[1]{%
2986   \@tenthstring{1}%
2987   \ifnum#1>0\relax
2988     -\@unitthstring{#1}%
2989   \fi
2990 }%
2991 \global\let\@@teenthstringportuges\@@teenthstringportuges

```

Hundreds:

```

2992 \newcommand*\@@hundredthstringportuges[1]{%
2993   \ifcase#1\relax
2994     \or cent\'esimo%
2995     \or ducent\'esimo%
2996     \or trecent\'esimo%
2997     \or quadringent\'esimo%
2998     \or q\'uingent\'esimo%
2999     \or seiscent\'esimo%
3000     \or setingent\'esimo%

```

```

3001 \or octingent\'esimo%
3002 \or nongent\'esimo%
3003 \fi
3004 }%
3005 \global\let\@@hundredthstringportuges\@@hundredthstringportuges

```

Units (feminine):

```

3006 \newcommand*\@@unitthstringFportuges [1] {%
3007 \ifcase#1\relax
3008 zero%
3009 \or primeira%
3010 \or segunda%
3011 \or terceira%
3012 \or quarta%
3013 \or quinta%
3014 \or sexta%
3015 \or s\'etima%
3016 \or oitava%
3017 \or nona%
3018 \fi
3019 }%
3020 \global\let\@@unitthstringFportuges\@@unitthstringFportuges

```

Tens (feminine):

```

3021 \newcommand*\@@tenthstringFportuges [1] {%
3022 \ifcase#1\relax
3023 \or d\'ecima%
3024 \or vig\'esima%
3025 \or trig\'esima%
3026 \or quadrag\'esima%
3027 \or q\'uinquag\'esima%
3028 \or sexag\'esima%
3029 \or setuag\'esima%
3030 \or octog\'esima%
3031 \or nonag\'esima%
3032 \fi
3033 }%
3034 \global\let\@@tenthstringFportuges\@@tenthstringFportuges

```

Hundreds (feminine):

```

3035 \newcommand*\@@hundredthstringFportuges [1] {%
3036 \ifcase#1\relax
3037 \or cent\'esima%
3038 \or ducent\'esima%
3039 \or trecent\'esima%
3040 \or quadringent\'esima%
3041 \or q\'uingent\'esima%
3042 \or seiscent\'esima%
3043 \or setingent\'esima%
3044 \or octingent\'esima%
3045 \or nongent\'esima%

```

```

3046 \fi
3047 }%
3048 \global\let\@@hundredthstringFportuges\@@hundredthstringFportuges

```

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

```

3049 \newcommand*\@@Unitthstringportuges[1]{%
3050 \ifcase#1\relax
3051 Zero%
3052 \or Primeiro%
3053 \or Segundo%
3054 \or Terceiro%
3055 \or Quarto%
3056 \or Quinto%
3057 \or Sexto%
3058 \or S\'etimo%
3059 \or Oitavo%
3060 \or Nono%
3061 \fi
3062 }%
3063 \global\let\@@Unitthstringportuges\@@Unitthstringportuges

```

Tens:

```

3064 \newcommand*\@@Tenthstringportuges[1]{%
3065 \ifcase#1\relax
3066 \or D\'ecimo%
3067 \or Vig\'esimo%
3068 \or Trig\'esimo%
3069 \or Quadrag\'esimo%
3070 \or Q\'uinquag\'esimo%
3071 \or Sexag\'esimo%
3072 \or Setuag\'esimo%
3073 \or Octog\'esimo%
3074 \or Nonag\'esimo%
3075 \fi
3076 }%
3077 \global\let\@@Tenthstringportuges\@@Tenthstringportuges

```

Hundreds:

```

3078 \newcommand*\@@Hundredthstringportuges[1]{%
3079 \ifcase#1\relax
3080 \or Cent\'esimo%
3081 \or Ducent\'esimo%
3082 \or Trecent\'esimo%
3083 \or Quadringent\'esimo%
3084 \or Q\'uingent\'esimo%
3085 \or Seiscent\'esimo%
3086 \or Setingent\'esimo%
3087 \or Octingent\'esimo%
3088 \or Nongent\'esimo%
3089 \fi
3090 }%

```

3091 \global\let\@@Hundredthstringportuges\@@Hundredthstringportuges

As above, but feminine. Units:

3092 \newcommand*\@@UnitthstringFportuges[1]{%

3093 \ifcase#1\relax

3094 Zera%

3095 \or Primeira%

3096 \or Segunda%

3097 \or Terceira%

3098 \or Quarta%

3099 \or Quinta%

3100 \or Sexta%

3101 \or S\'etima%

3102 \or Oitava%

3103 \or Nona%

3104 \fi

3105 }%

3106 \global\let\@@UnitthstringFportuges\@@UnitthstringFportuges

Tens (feminine);

3107 \newcommand*\@@TenthstringFportuges[1]{%

3108 \ifcase#1\relax

3109 \or D\'ecima%

3110 \or Vig\'esima%

3111 \or Trig\'esima%

3112 \or Quadrag\'esima%

3113 \or Q\'uinquag\'esima%

3114 \or Sexag\'esima%

3115 \or Setuag\'esima%

3116 \or Octog\'esima%

3117 \or Nonag\'esima%

3118 \fi

3119 }%

3120 \global\let\@@TenthstringFportuges\@@TenthstringFportuges

Hundreds (feminine):

3121 \newcommand*\@@HundredthstringFportuges[1]{%

3122 \ifcase#1\relax

3123 \or Cent\'esima%

3124 \or Ducent\'esima%

3125 \or Trecent\'esima%

3126 \or Quadringent\'esima%

3127 \or Q\'uingent\'esima%

3128 \or Seiscent\'esima%

3129 \or Setingent\'esima%

3130 \or Octingent\'esima%

3131 \or Nongent\'esima%

3132 \fi

3133 }%

3134 \global\let\@@HundredthstringFportuges\@@HundredthstringFportuges

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

```

3135 \newcommand*\@@numberstringportuges[2]{%
3136 \ifnum#1>99999\relax
3137 \PackageError{fmtcount}{Out of range}%
3138 {This macro only works for values less than 100000}%
3139 \else
3140 \ifnum#1<0\relax
3141 \PackageError{fmtcount}{Negative numbers not permitted}%
3142 {This macro does not work for negative numbers, however
3143 you can try typing "minus" first, and then pass the modulus of
3144 this number}%
3145 \fi
3146 \fi
3147 \def#2{}%
3148 \@strctr=#1\relax \divide\@strctr by 1000\relax
3149 \ifnum\@strctr>9\relax
    #1 is greater or equal to 10000
3150 \divide\@strctr by 10\relax
3151 \ifnum\@strctr>1\relax
3152 \let\@@fc@numstr#2\relax
3153 \protected@edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
3154 \@strctr=#1 \divide\@strctr by 1000\relax
3155 \@FCmodulo{\@strctr}{10}%
3156 \ifnum\@strctr>0
3157 \ifnum\@strctr=1\relax
3158 \let\@@fc@numstr#2\relax
3159 \protected@edef#2{\@@fc@numstr\ \@andname}%
3160 \fi
3161 \let\@@fc@numstr#2\relax
3162 \protected@edef#2{\@@fc@numstr\ \@unitstring{\@strctr}}%
3163 \fi
3164 \else
3165 \@strctr=#1\relax
3166 \divide\@strctr by 1000\relax
3167 \@FCmodulo{\@strctr}{10}%
3168 \let\@@fc@numstr#2\relax
3169 \protected@edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
3170 \fi
3171 \let\@@fc@numstr#2\relax
3172 \protected@edef#2{\@@fc@numstr\ \@thousand}%
3173 \else
3174 \ifnum\@strctr>0\relax
3175 \ifnum\@strctr>1\relax
3176 \let\@@fc@numstr#2\relax
3177 \protected@edef#2{\@@fc@numstr\@unitstring{\@strctr}\ }%

```

```

3178 \fi
3179 \let\@fc@numstr#2\relax
3180 \protected@edef#2{\@fc@numstr\@thousand}%
3181 \fi
3182 \fi
3183 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
3184 \divide\@strctr by 100\relax
3185 \ifnum\@strctr>0\relax
3186 \ifnum#1>1000 \relax
3187 \let\@fc@numstr#2\relax
3188 \protected@edef#2{\@fc@numstr\ }%
3189 \fi
3190 \@tmpstrctr=#1\relax
3191 \@FCmodulo{\@tmpstrctr}{1000}%
3192 \let\@fc@numstr#2\relax
3193 \ifnum\@tmpstrctr=100\relax
3194 \protected@edef#2{\@fc@numstr\@tenstring{10}}%
3195 \else
3196 \protected@edef#2{\@fc@numstr\@hundredstring{\@strctr}}%
3197 \fi%
3198 \fi
3199 \@strctr=#1\relax \@FCmodulo{\@strctr}{100}%
3200 \ifnum#1>100\relax
3201 \ifnum\@strctr>0\relax
3202 \let\@fc@numstr#2\relax
3203 \protected@edef#2{\@fc@numstr\ \@andname\ }%
3204 \fi
3205 \fi
3206 \ifnum\@strctr>19\relax
3207 \divide\@strctr by 10\relax
3208 \let\@fc@numstr#2\relax
3209 \protected@edef#2{\@fc@numstr\@tenstring{\@strctr}}%
3210 \@strctr=#1\relax \@FCmodulo{\@strctr}{10}%
3211 \ifnum\@strctr>0
3212 \ifnum\@strctr=1\relax
3213 \let\@fc@numstr#2\relax
3214 \protected@edef#2{\@fc@numstr\ \@andname}%
3215 \else
3216 \ifnum#1>100\relax
3217 \let\@fc@numstr#2\relax
3218 \protected@edef#2{\@fc@numstr\ \@andname}%
3219 \fi
3220 \fi
3221 \let\@fc@numstr#2\relax
3222 \protected@edef#2{\@fc@numstr\ \@unitstring{\@strctr}}%
3223 \fi
3224 \else
3225 \ifnum\@strctr<10\relax
3226 \ifnum\@strctr=0\relax

```

```

3227     \ifnum#1<100\relax
3228         \let\@fc@numstr#2\relax
3229         \protected@edef#2{\@fc@numstr\@unitstring{\@strctr}}%
3230     \fi
3231 \else %(>0,<10)
3232     \let\@fc@numstr#2\relax
3233     \protected@edef#2{\@fc@numstr\@unitstring{\@strctr}}%
3234 \fi
3235 \else%>10
3236     \@FCmodulo{\@strctr}{10}%
3237     \let\@fc@numstr#2\relax
3238     \protected@edef#2{\@fc@numstr\@teenstring{\@strctr}}%
3239 \fi
3240 \fi
3241 }%
3242 \global\let\@numberstringportuges\@numberstringportuges

```

As above, but for ordinals.

```

3243 \newcommand*\@ordinalstringportuges [2] {%
3244 \@strctr=#1\relax
3245 \ifnum#1>99999
3246 \PackageError{fmtcount}{Out of range}%
3247 {This macro only works for values less than 100000}%
3248 \else
3249 \ifnum#1<0
3250 \PackageError{fmtcount}{Negative numbers not permitted}%
3251 {This macro does not work for negative numbers, however
3252 you can try typing "minus" first, and then pass the modulus of
3253 this number}%
3254 \else
3255 \def#2{}%
3256 \ifnum\@strctr>999\relax
3257     \divide\@strctr by 1000\relax
3258     \ifnum\@strctr>1\relax
3259         \ifnum\@strctr>9\relax
3260             \@tmpstrctr=\@strctr
3261             \ifnum\@strctr<20
3262                 \@FCmodulo{\@tmpstrctr}{10}%
3263                 \let\@fc@ordstr#2\relax
3264                 \protected@edef#2{\@fc@ordstr\@teenthstring{\@tmpstrctr}}%
3265             \else
3266                 \divide\@tmpstrctr by 10\relax
3267                 \let\@fc@ordstr#2\relax
3268                 \protected@edef#2{\@fc@ordstr\@tenthstring{\@tmpstrctr}}%
3269                 \@tmpstrctr=\@strctr
3270                 \@FCmodulo{\@tmpstrctr}{10}%
3271                 \ifnum\@tmpstrctr>0\relax
3272                     \let\@fc@ordstr#2\relax
3273                     \protected@edef#2{\@fc@ordstr\@unitthstring{\@tmpstrctr}}%
3274                 \fi

```



```

3275     \fi
3276   \else
3277     \let\@@fc@ordstr#2\relax
3278     \protected@edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
3279   \fi
3280 \fi
3281 \let\@@fc@ordstr#2\relax
3282 \protected@edef#2{\@@fc@ordstr\@thousandth}%
3283 \fi
3284 \@strctr=#1\relax
3285 \@FCmodulo{\@strctr}{1000}%
3286 \ifnum\@strctr>99\relax
3287   \@tmpstrctr=\@strctr
3288   \divide\@tmpstrctr by 100\relax
3289   \ifnum#1>1000\relax
3290     \let\@@fc@ordstr#2\relax
3291     \protected@edef#2{\@@fc@ordstr-}%
3292   \fi
3293   \let\@@fc@ordstr#2\relax
3294   \protected@edef#2{\@@fc@ordstr\@hundredthstring{\@tmpstrctr}}%
3295 \fi
3296 \@FCmodulo{\@strctr}{100}%
3297 \ifnum#1>99\relax
3298   \ifnum\@strctr>0\relax
3299     \let\@@fc@ordstr#2\relax
3300     \protected@edef#2{\@@fc@ordstr-}%
3301   \fi
3302 \fi
3303 \ifnum\@strctr>9\relax
3304   \@tmpstrctr=\@strctr
3305   \divide\@tmpstrctr by 10\relax
3306   \let\@@fc@ordstr#2\relax
3307   \protected@edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
3308   \@tmpstrctr=\@strctr
3309   \@FCmodulo{\@tmpstrctr}{10}%
3310   \ifnum\@tmpstrctr>0\relax
3311     \let\@@fc@ordstr#2\relax
3312     \protected@edef#2{\@@fc@ordstr-\@unitthstring{\@tmpstrctr}}%
3313   \fi
3314 \else
3315   \ifnum\@strctr=0\relax
3316     \ifnum#1=0\relax
3317       \let\@@fc@ordstr#2\relax
3318       \protected@edef#2{\@@fc@ordstr\@unitstring{0}}%
3319     \fi
3320   \else
3321     \let\@@fc@ordstr#2\relax
3322     \protected@edef#2{\@@fc@ordstr\@unitthstring{\@strctr}}%
3323   \fi

```

```

3324 \fi
3325 \fi
3326 \fi
3327 }%
3328 \global\let\@@ordinalstringportuges\@@ordinalstringportuges

```

10.1.15 fc-portuguese.def

```
3329 \ProvidesFCLanguage{portuguese}[2014/06/09]%
```

Load fc-portuges.def if not already loaded.

```
3330 \FCloadlang{portuges}%
```

Set |portuguese| to be equivalent to |portuges|.

```

3331 \global\let\@ordinalMportuguese=\@ordinalMportuges
3332 \global\let\@ordinalFportuguese=\@ordinalFportuges
3333 \global\let\@ordinalNportuguese=\@ordinalNportuges
3334 \global\let\@numberstringMportuguese=\@numberstringMportuges
3335 \global\let\@numberstringFportuguese=\@numberstringFportuges
3336 \global\let\@numberstringNportuguese=\@numberstringNportuges
3337 \global\let\@NumberstringMportuguese=\@NumberstringMportuges
3338 \global\let\@NumberstringFportuguese=\@NumberstringFportuges
3339 \global\let\@NumberstringNportuguese=\@NumberstringNportuges
3340 \global\let\@ordinalstringMportuguese=\@ordinalstringMportuges
3341 \global\let\@ordinalstringFportuguese=\@ordinalstringFportuges
3342 \global\let\@ordinalstringNportuguese=\@ordinalstringNportuges
3343 \global\let\@OrdinalstringMportuguese=\@OrdinalstringMportuges
3344 \global\let\@OrdinalstringFportuguese=\@OrdinalstringFportuges
3345 \global\let\@OrdinalstringNportuguese=\@OrdinalstringNportuges

```

10.1.16 fc-spanish.def

Spanish definitions

```
3346 \ProvidesFCLanguage{spanish}[2016/01/12]%
```

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:

```

3347 \newcommand*\@ordinalMspanish[2]{%
3348   \edef#2{\number#1\relax\noexpand\fmtord{o}}%
3349 }%
3350 \global\let\@ordinalMspanish\@ordinalMspanish

```

Feminine:

```

3351 \newcommand*\@ordinalFspanish[2]{%
3352   \edef#2{\number#1\relax\noexpand\fmtord{a}}%
3353 }%
3354 \global\let\@ordinalFspanish\@ordinalFspanish

```

Make neuter same as masculine:

```
3355 \global\let\@ordinalNspanish\@ordinalMspanish
```

Convert a number to text. The easiest way to do this is to break it up into units, tens, teens, twenties and hundreds. Units (argument must be a number from 0 to 9):

```
3356 \newcommand*\@@unitstringspanish[1]{%
3357   \ifcase#1\relax
3358     cero%
3359     \or uno%
3360     \or dos%
3361     \or tres%
3362     \or cuatro%
3363     \or cinco%
3364     \or seis%
3365     \or siete%
3366     \or ocho%
3367     \or nueve%
3368   \fi
3369 }%
3370 \global\let\@@unitstringspanish\@@unitstringspanish
```

Feminine:

```
3371 \newcommand*\@@unitstringFspanish[1]{%
3372   \ifcase#1\relax
3373     cera%
3374     \or una%
3375     \or dos%
3376     \or tres%
3377     \or cuatro%
3378     \or cinco%
3379     \or seis%
3380     \or siete%
3381     \or ocho%
3382     \or nueve%
3383   \fi
3384 }%
3385 \global\let\@@unitstringFspanish\@@unitstringFspanish
```

Tens (argument must go from 1 to 10):

```
3386 \newcommand*\@@tenstringspanish[1]{%
3387   \ifcase#1\relax
3388     \or diez%
3389     \or veinte%
3390     \or treinta%
3391     \or cuarenta%
3392     \or cincuenta%
3393     \or sesenta%
3394     \or setenta%
3395     \or ochenta%
3396     \or noventa%
3397     \or cien%
3398   \fi
3399 }%
```

```

3400 \global\let\@@tenstringspanish\@@tenstringspanish
    Teens:
3401 \newcommand*\@@teenstringspanish[1]{%
3402   \ifcase#1\relax
3403     diez%
3404     \or once%
3405     \or doce%
3406     \or trece%
3407     \or catorce%
3408     \or quince%
3409     \or dieciséis%
3410     \or diecisiete%
3411     \or dieciocho%
3412     \or diecinueve%
3413   \fi
3414 }%
3415 \global\let\@@teenstringspanish\@@teenstringspanish
    Twenties:
3416 \newcommand*\@@twentystringspanish[1]{%
3417   \ifcase#1\relax
3418     veinte%
3419     \or veintiuno%
3420     \or veintidós%
3421     \or veintitrés%
3422     \or veinticuatro%
3423     \or veinticinco%
3424     \or veintiséis%
3425     \or veintisiete%
3426     \or veintiocho%
3427     \or veintinueve%
3428   \fi
3429 }%
3430 \global\let\@@twentystringspanish\@@twentystringspanish
    Feminine form:
3431 \newcommand*\@@twentystringFspanish[1]{%
3432   \ifcase#1\relax
3433     veinte%
3434     \or veintiuna%
3435     \or veintidós%
3436     \or veintitrés%
3437     \or veinticuatro%
3438     \or veinticinco%
3439     \or veintiséis%
3440     \or veintisiete%
3441     \or veintiocho%
3442     \or veintinueve%
3443   \fi
3444 }%

```

3445 \global\let\@@twentystringFspanish\@@twentystringFspanish

Hundreds:

3446 \newcommand*\@@hundredstringspanish[1]{%

3447 \ifcase#1\relax

3448 \or ciento%

3449 \or doscientos%

3450 \or trescientos%

3451 \or cuatrocientos%

3452 \or quinientos%

3453 \or seiscientos%

3454 \or setecientos%

3455 \or ochocientos%

3456 \or novecientos%

3457 \fi

3458 }%

3459 \global\let\@@hundredstringspanish\@@hundredstringspanish

Feminine form:

3460 \newcommand*\@@hundredstringFspanish[1]{%

3461 \ifcase#1\relax

3462 \or cienta%

3463 \or doscientas%

3464 \or trescientas%

3465 \or cuatrocientas%

3466 \or quinientas%

3467 \or seiscientas%

3468 \or setecientas%

3469 \or ochocientas%

3470 \or novecientas%

3471 \fi

3472 }%

3473 \global\let\@@hundredstringFspanish\@@hundredstringFspanish

As above, but with initial letter uppercase:

3474 \newcommand*\@@Unitstringspanish[1]{%

3475 \ifcase#1\relax

3476 Cero%

3477 \or Uno%

3478 \or Dos%

3479 \or Tres%

3480 \or Cuatro%

3481 \or Cinco%

3482 \or Seis%

3483 \or Siete%

3484 \or Ocho%

3485 \or Nueve%

3486 \fi

3487 }%

3488 \global\let\@@Unitstringspanish\@@Unitstringspanish

Feminine form:

```
3489 \newcommand*\@@UnitstringFspanish[1]{%
3490   \ifcase#1\relax
3491     Cera%
3492     \or Una%
3493     \or Dos%
3494     \or Tres%
3495     \or Cuatro%
3496     \or Cinco%
3497     \or Seis%
3498     \or Siete%
3499     \or Ocho%
3500     \or Nueve%
3501   \fi
3502 }%
3503 \global\let\@@UnitstringFspanish\@@UnitstringFspanish
```

Tens:

```
3504 %\changes{2.0}{2012-06-18}{fixed spelling mistake (correction
3505 %provided by Fernando Maldonado)}
3506 \newcommand*\@@Tenstringspanish[1]{%
3507   \ifcase#1\relax
3508     \or Diez%
3509     \or Veinte%
3510     \or Treinta%
3511     \or Cuarenta%
3512     \or Cincuenta%
3513     \or Sesenta%
3514     \or Setenta%
3515     \or Ochenta%
3516     \or Noventa%
3517     \or Cien%
3518   \fi
3519 }%
3520 \global\let\@@Tenstringspanish\@@Tenstringspanish
```

Teens:

```
3521 \newcommand*\@@Teenstringspanish[1]{%
3522   \ifcase#1\relax
3523     Diez%
3524     \or Once%
3525     \or Doce%
3526     \or Trece%
3527     \or Catorce%
3528     \or Quince%
3529     \or Dieciséis%
3530     \or Diecisiete%
3531     \or Dieciocho%
3532     \or Diecinueve%
3533   \fi
```

```

3534 }%
3535 \global\let\@@Teenstringspanish\@@Teenstringspanish
    Twenties:
3536 \newcommand*\@@Twentystringspanish[1]{%
3537   \ifcase#1\relax
3538     Veinte%
3539     \or Veintiuno%
3540     \or Veintidós%
3541     \or Veintitrés%
3542     \or Veinticuatro%
3543     \or Veinticinco%
3544     \or Veintiséis%
3545     \or Veintisiete%
3546     \or Veintiocho%
3547     \or Veintinueve%
3548   \fi
3549 }%
3550 \global\let\@@Twentystringspanish\@@Twentystringspanish
    Feminine form:
3551 \newcommand*\@@TwentystringFspanish[1]{%
3552   \ifcase#1\relax
3553     Veinte%
3554     \or Veintiuna%
3555     \or Veintidós%
3556     \or Veintitrés%
3557     \or Veinticuatro%
3558     \or Veinticinco%
3559     \or Veintiséis%
3560     \or Veintisiete%
3561     \or Veintiocho%
3562     \or Veintinueve%
3563   \fi
3564 }%
3565 \global\let\@@TwentystringFspanish\@@TwentystringFspanish
    Hundreds:
3566 \newcommand*\@@Hundredstringspanish[1]{%
3567   \ifcase#1\relax
3568     \or Ciento%
3569     \or Doscientos%
3570     \or Trescientos%
3571     \or Cuatrocientos%
3572     \or Quinientos%
3573     \or Seiscientos%
3574     \or Setecientos%
3575     \or Ochocientos%
3576     \or Novecientos%
3577   \fi
3578 }%

```

```
3579 \global\let\@@Hundredstringspanish\@@Hundredstringspanish
```

Feminine form:

```
3580 \newcommand*\@@HundredstringFspanish[1]{%
3581   \ifcase#1\relax
3582     \or Cienta%
3583     \or Doscientas%
3584     \or Trescientas%
3585     \or Cuatrocientas%
3586     \or Quinientas%
3587     \or Seiscientas%
3588     \or Setecientas%
3589     \or Ochocientas%
3590     \or Novecientas%
3591   \fi
3592 }%
```

```
3593 \global\let\@@HundredstringFspanish\@@HundredstringFspanish
```

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

```
3594 \newcommand*\@numberstringMspanish[2]{%
3595   \let\@unitstring=\@unitstringspanish
3596   \let\@teenstring=\@teenstringspanish
3597   \let\@tenstring=\@tenstringspanish
3598   \let\@twentystring=\@twentystringspanish
3599   \let\@hundredstring=\@hundredstringspanish
3600   \def\@hundred{cien}\def\@thousand{mil}%
3601   \def\@andname{y}%
3602   \@numberstringspanish{#1}{#2}%
3603 }%
```

```
3604 \global\let\@numberstringMspanish\@numberstringMspanish
```

Feminine form:

```
3605 \newcommand*\@numberstringFspanish[2]{%
3606   \let\@unitstring=\@unitstringFspanish
3607   \let\@teenstring=\@teenstringspanish
3608   \let\@tenstring=\@tenstringspanish
3609   \let\@twentystring=\@twentystringFspanish
3610   \let\@hundredstring=\@hundredstringFspanish
3611   \def\@hundred{cien}\def\@thousand{mil}%
3612   \def\@andname{b}%
3613   \@numberstringspanish{#1}{#2}%
3614 }%
```

```
3615 \global\let\@numberstringFspanish\@numberstringFspanish
```

Make neuter same as masculine:

```
3616 \global\let\@numberstringNspanish\@numberstringMspanish
```

As above, but initial letters in upper case:

```
3617 \newcommand*\@NumberstringMspanish[2]{%
```



```

3618 \let\@unitstring=\@Unitstringspanish
3619 \let\@teenstring=\@Teenstringspanish
3620 \let\@tenstring=\@Tenstringspanish
3621 \let\@twentystring=\@Twentystringspanish
3622 \let\@hundredstring=\@Hundredstringspanish
3623 \def\@andname{y}%
3624 \def\@hundred{Cien}\def\@thousand{Mil}%
3625 \@numberstringspanish{#1}{#2}%
3626 }%
3627 \global\let\@NumberstringMspanish\@NumberstringMspanish

  Feminine form:
3628 \newcommand*\@NumberstringFspanish}[2]{%
3629 \let\@unitstring=\@UnitstringFspanish
3630 \let\@teenstring=\@Teenstringspanish
3631 \let\@tenstring=\@Tenstringspanish
3632 \let\@twentystring=\@TwentystringFspanish
3633 \let\@hundredstring=\@HundredstringFspanish
3634 \def\@andname{b}%
3635 \def\@hundred{Cien}\def\@thousand{Mil}%
3636 \@numberstringspanish{#1}{#2}%
3637 }%
3638 \global\let\@NumberstringFspanish\@NumberstringFspanish

  Make neuter same as masculine:
3639 \global\let\@NumberstringNspanish\@NumberstringMspanish

  As above, but for ordinals.
3640 \newcommand*\@OrdinalstringMspanish}[2]{%
3641 \let\@unitthstring=\@Unitthstringspanish
3642 \let\@unitstring=\@Unitstringspanish
3643 \let\@teenthstring=\@Teenthstringspanish
3644 \let\@tenthstring=\@Tenthstringspanish
3645 \let\@hundredthstring=\@Hundredthstringspanish
3646 \def\@thousandth{milésimo}%
3647 \@ordinalstringspanish{#1}{#2}%
3648 }%
3649 \global\let\@OrdinalstringMspanish\@OrdinalstringMspanish

  Feminine form:
3650 \newcommand*\@OrdinalstringFspanish}[2]{%
3651 \let\@unitthstring=\@UnitthstringFspanish
3652 \let\@unitstring=\@UnitstringFspanish
3653 \let\@teenthstring=\@TeenthstringFspanish
3654 \let\@tenthstring=\@TenthstringFspanish
3655 \let\@hundredthstring=\@HundredthstringFspanish
3656 \def\@thousandth{milésima}%
3657 \@ordinalstringspanish{#1}{#2}%
3658 }%
3659 \global\let\@OrdinalstringFspanish\@OrdinalstringFspanish

```

Make neuter same as masculine:

```
3660 \global\let\@ordinalstringNspanish\@ordinalstringMspanish
```

As above, but with initial letters in upper case.

```
3661 \newcommand*\@OrdinalstringMspanish}[2]{%
3662   \let\@unitthstring=\@Unitthstringspanish
3663   \let\@unitstring=\@Unitstringspanish
3664   \let\@teenthstring=\@Teenthstringspanish
3665   \let\@tenthstring=\@Tenthstringspanish
3666   \let\@hundredthstring=\@Hundredthstringspanish
3667   \def\@thousandth{Milésimo}%
3668   \@ordinalstringspanish{#1}{#2}%
3669 }
3670 \global\let\@OrdinalstringMspanish\@OrdinalstringMspanish
```

Feminine form:

```
3671 \newcommand*\@OrdinalstringFspanish}[2]{%
3672   \let\@unitthstring=\@UnitthstringFspanish
3673   \let\@unitstring=\@UnitstringFspanish
3674   \let\@teenthstring=\@TeenthstringFspanish
3675   \let\@tenthstring=\@TenthstringFspanish
3676   \let\@hundredthstring=\@HundredthstringFspanish
3677   \def\@thousandth{Milésima}%
3678   \@ordinalstringspanish{#1}{#2}%
3679 }%
3680 \global\let\@OrdinalstringFspanish\@OrdinalstringFspanish
```

Make neuter same as masculine:

```
3681 \global\let\@OrdinalstringNspanish\@OrdinalstringMspanish
```

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

```
3682 \newcommand*\@unitthstringspanish[1]{%
3683   \ifcase#1\relax
3684     cero%
3685     \or primero%
3686     \or segundo%
3687     \or tercero%
3688     \or cuarto%
3689     \or quinto%
3690     \or sexto%
3691     \or séptimo%
3692     \or octavo%
3693     \or noveno%
3694   \fi
3695 }%
3696 \global\let\@unitthstringspanish\@unitthstringspanish
```

Tens:

```
3697 \newcommand*\@tenthstringspanish[1]{%
3698   \ifcase#1\relax
```

3699 \or décimo%
 3700 \or vigésimo%
 3701 \or trigésimo%
 3702 \or cuadragésimo%
 3703 \or quincuagésimo%
 3704 \or sexagésimo%
 3705 \or septuagésimo%
 3706 \or octogésimo%
 3707 \or nonagésimo%
 3708 \fi
 3709 }%
 3710 \global\let\@@tenthstringspanish\@@tenthstringspanish

Teens:

3711 \newcommand*\@@teenthstringspanish[1]{%
 3712 \ifcase#1\relax
 3713 décimo%
 3714 \or undécimo%
 3715 \or duodécimo%
 3716 \or decimotercero%
 3717 \or decimocuarto%
 3718 \or decimoquinto%
 3719 \or decimosexto%
 3720 \or decimoséptimo%
 3721 \or decimooctavo%
 3722 \or decimonoveno%
 3723 \fi
 3724 }%
 3725 \global\let\@@teenthstringspanish\@@teenthstringspanish

Hundreds:

3726 \newcommand*\@@hundredthstringspanish[1]{%
 3727 \ifcase#1\relax
 3728 \or centésimo%
 3729 \or ducentésimo%
 3730 \or tricentésimo%
 3731 \or cuadringentésimo%
 3732 \or quingentésimo%
 3733 \or sexcentésimo%
 3734 \or septingésimo%
 3735 \or octingentésimo%
 3736 \or noningentésimo%
 3737 \fi
 3738 }%
 3739 \global\let\@@hundredthstringspanish\@@hundredthstringspanish

Units (feminine):

3740 \newcommand*\@@unitthstringFspanish[1]{%
 3741 \ifcase#1\relax
 3742 cera%
 3743 \or primera%

3744 \or segunda%
 3745 \or tercera%
 3746 \or cuarta%
 3747 \or quinta%
 3748 \or sexta%
 3749 \or séptima%
 3750 \or octava%
 3751 \or novena%
 3752 \fi
 3753 }%
 3754 \global\let\@@unitthstringFspanish\@@unitthstringFspanish

Tens (feminine):

3755 \newcommand*\@@tenthstringFspanish[1]{%
 3756 \ifcase#1\relax
 3757 \or décima%
 3758 \or vigésima%
 3759 \or trigésima%
 3760 \or cuadragésima%
 3761 \or quincuagésima%
 3762 \or sexagésima%
 3763 \or septuagésima%
 3764 \or octogésima%
 3765 \or nonagésima%
 3766 \fi
 3767 }%
 3768 \global\let\@@tenthstringFspanish\@@tenthstringFspanish

Teens (feminine)

3769 \newcommand*\@@teenthstringFspanish[1]{%
 3770 \ifcase#1\relax
 3771 décima%
 3772 \or undécima%
 3773 \or duodécima%
 3774 \or decimotercera%
 3775 \or decimocuarta%
 3776 \or decimoquinta%
 3777 \or decimosexta%
 3778 \or decimoséptima%
 3779 \or decimoctava%
 3780 \or decimonovena%
 3781 \fi
 3782 }%
 3783 \global\let\@@teenthstringFspanish\@@teenthstringFspanish

Hundreds (feminine)

3784 \newcommand*\@@hundredthstringFspanish[1]{%
 3785 \ifcase#1\relax
 3786 \or centésima%
 3787 \or ducentésima%
 3788 \or tricentésima%

```

3789 \or cuadingentésima%
3790 \or quingentésima%
3791 \or sexcentésima%
3792 \or septingésima%
3793 \or octingentésima%
3794 \or noningentésima%
3795 \fi
3796 }%
3797 \global\let\@@hundredthstringFspanish\@@hundredthstringFspanish

```

As above, but with initial letters in upper case

```

3798 \newcommand*\@@Unitthstringspanish[1]{%
3799 \ifcase#1\relax
3800 Cero%
3801 \or Primero%
3802 \or Segundo%
3803 \or Tercero%
3804 \or Cuarto%
3805 \or Quinto%
3806 \or Sexto%
3807 \or Séptimo%
3808 \or Octavo%
3809 \or Noveno%
3810 \fi
3811 }%
3812 \global\let\@@Unitthstringspanish\@@Unitthstringspanish

```

Tens:

```

3813 \newcommand*\@@Tenthstringspanish[1]{%
3814 \ifcase#1\relax
3815 \or Décimo%
3816 \or Vigésimo%
3817 \or Trigésimo%
3818 \or Cuadragésimo%
3819 \or Quincuagésimo%
3820 \or Sexagésimo%
3821 \or Septuagésimo%
3822 \or Octogésimo%
3823 \or Nonagésimo%
3824 \fi
3825 }%
3826 \global\let\@@Tenthstringspanish\@@Tenthstringspanish

```

Teens:

```

3827 \newcommand*\@@Teenthstringspanish[1]{%
3828 \ifcase#1\relax
3829 Décimo%
3830 \or Undécimo%
3831 \or Duodécimo%
3832 \or Decimotercero%
3833 \or Decimocuarto%

```

3834 \or Decimoquinto%
 3835 \or Decimosexto%
 3836 \or Decimoséptimo%
 3837 \or Decimooctavo%
 3838 \or Decimonoveno%
 3839 \fi
 3840 }%
 3841 \global\let\@@Teenthstringspanish\@@Teenthstringspanish

Hundreds

3842 \newcommand*\@@Hundredthstringspanish[1]{%
 3843 \ifcase#1\relax
 3844 \or Centésimo%
 3845 \or Ducentésimo%
 3846 \or Tricentésimo%
 3847 \or Cuadringentésimo%
 3848 \or Quingentésimo%
 3849 \or Sexcentésimo%
 3850 \or Septingésimo%
 3851 \or Octingentésimo%
 3852 \or Noningentésimo%
 3853 \fi
 3854 }%
 3855 \global\let\@@Hundredthstringspanish\@@Hundredthstringspanish

As above, but feminine.

3856 \newcommand*\@@UnitthstringFspanish[1]{%
 3857 \ifcase#1\relax
 3858 Cera%
 3859 \or Primera%
 3860 \or Segunda%
 3861 \or Tercera%
 3862 \or Cuarta%
 3863 \or Quinta%
 3864 \or Sexta%
 3865 \or Séptima%
 3866 \or Octava%
 3867 \or Novena%
 3868 \fi
 3869 }%
 3870 \global\let\@@UnitthstringFspanish\@@UnitthstringFspanish

Tens (feminine)

3871 \newcommand*\@@TenthstringFspanish[1]{%
 3872 \ifcase#1\relax
 3873 \or Décima%
 3874 \or Vigésima%
 3875 \or Trigésima%
 3876 \or Cuadragésima%
 3877 \or Quincuagésima%
 3878 \or Sexagésima%

```

3879 \or Septuagésima%
3880 \or Octogésima%
3881 \or Nonagésima%
3882 \fi
3883 }%
3884 \global\let\@@TenthstringFspanish\@@TenthstringFspanish

```

Teens (feminine):

```

3885 \newcommand*\@@TeenthstringFspanish[1]{%
3886 \ifcase#1\relax
3887 Décima%
3888 \or Undécima%
3889 \or Duodécima%
3890 \or Decimotercera%
3891 \or Decimocuarta%
3892 \or Decimoquinta%
3893 \or Decimosexta%
3894 \or Decimoséptima%
3895 \or Decimoctava%
3896 \or Decimonovena%
3897 \fi
3898 }%
3899 \global\let\@@TeenthstringFspanish\@@TeenthstringFspanish

```

Hundreds (feminine):

```

3900 \newcommand*\@@HundredthstringFspanish[1]{%
3901 \ifcase#1\relax
3902 \or Centésima%
3903 \or Ducentésima%
3904 \or Tricentésima%
3905 \or Cuadringentésima%
3906 \or Quingentésima%
3907 \or Sexcentésima%
3908 \or Septingésima%
3909 \or Octingentésima%
3910 \or Noningentésima%
3911 \fi
3912 }%
3913 \global\let\@@HundredthstringFspanish\@@HundredthstringFspanish

```

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

```

3914 \newcommand*\@@numberstringspanish[2]{%
3915 \ifnum#1>99999
3916 \PackageError{fmtcount}{Out of range}%
3917 {This macro only works for values less than 100000}%
3918 \else
3919 \ifnum#1<0

```

```

3920 \PackageError{fmtcount}{Negative numbers not permitted}%
3921 {This macro does not work for negative numbers, however
3922 you can try typing "minus" first, and then pass the modulus of
3923 this number}%
3924 \fi
3925 \fi
3926 \def#2{}%
3927 \@strctr=#1\relax \divide\@strctr by 1000\relax
3928 \ifnum\@strctr>9
    #1 is greater or equal to 10000
3929 \divide\@strctr by 10
3930 \ifnum\@strctr>1
3931 \let\@fc@numstr#2\relax
3932 \edef#2{\@fc@numstr\@tenstring{\@strctr}}%
3933 \@strctr=#1 \divide\@strctr by 1000\relax
3934 \@FCmodulo{\@strctr}{10}%
3935 \ifnum\@strctr>0\relax
3936 \let\@fc@numstr#2\relax
3937 \edef#2{\@fc@numstr\@andname\@unitstring{\@strctr}}%
3938 \fi
3939 \else
3940 \@strctr=#1\relax
3941 \divide\@strctr by 1000\relax
3942 \@FCmodulo{\@strctr}{10}%
3943 \let\@fc@numstr#2\relax
3944 \edef#2{\@fc@numstr\@teenstring{\@strctr}}%
3945 \fi
3946 \let\@fc@numstr#2\relax
3947 \edef#2{\@fc@numstr\@thousand}%
3948 \else
3949 \ifnum\@strctr>0\relax
3950 \ifnum\@strctr>1\relax
3951 \let\@fc@numstr#2\relax
3952 \edef#2{\@fc@numstr\@unitstring{\@strctr}\ }%
3953 \fi
3954 \let\@fc@numstr#2\relax
3955 \edef#2{\@fc@numstr\@thousand}%
3956 \fi
3957 \fi
3958 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
3959 \divide\@strctr by 100\relax
3960 \ifnum\@strctr>0\relax
3961 \ifnum#1>1000\relax
3962 \let\@fc@numstr#2\relax
3963 \edef#2{\@fc@numstr\ }%
3964 \fi
3965 \@tmpstrctr=#1\relax
3966 \@FCmodulo{\@tmpstrctr}{1000}%
3967 \ifnum\@tmpstrctr=100\relax

```



```

3968 \let\@fc@numstr#2\relax
3969 \edef#2{\@fc@numstr\@tenstring{10}}%
3970 \else
3971 \let\@fc@numstr#2\relax
3972 \edef#2{\@fc@numstr\@hundredstring{\@strctr}}%
3973 \fi
3974 \fi
3975 \@strctr=#1\relax \@FCmodulo{\@strctr}{100}%
3976 \ifnum#1>100\relax
3977 \ifnum\@strctr>0\relax
3978 \let\@fc@numstr#2\relax
3979 \edef#2{\@fc@numstr\ }%
3980 \fi
3981 \fi
3982 \ifnum\@strctr>29\relax
3983 \divide\@strctr by 10\relax
3984 \let\@fc@numstr#2\relax
3985 \edef#2{\@fc@numstr\@tenstring{\@strctr}}%
3986 \@strctr=#1\relax \@FCmodulo{\@strctr}{10}%
3987 \ifnum\@strctr>0\relax
3988 \let\@fc@numstr#2\relax
3989 \edef#2{\@fc@numstr\ \@andname\ \@unitstring{\@strctr}}%
3990 \fi
3991 \else
3992 \ifnum\@strctr<10\relax
3993 \ifnum\@strctr=0\relax
3994 \ifnum#1<100\relax
3995 \let\@fc@numstr#2\relax
3996 \edef#2{\@fc@numstr\@unitstring{\@strctr}}%
3997 \fi
3998 \else
3999 \let\@fc@numstr#2\relax
4000 \edef#2{\@fc@numstr\@unitstring{\@strctr}}%
4001 \fi
4002 \else
4003 \ifnum\@strctr>19\relax
4004 \@FCmodulo{\@strctr}{10}%
4005 \let\@fc@numstr#2\relax
4006 \edef#2{\@fc@numstr\@twentystring{\@strctr}}%
4007 \else
4008 \@FCmodulo{\@strctr}{10}%
4009 \let\@fc@numstr#2\relax
4010 \edef#2{\@fc@numstr\@teenstring{\@strctr}}%
4011 \fi
4012 \fi
4013 \fi
4014 }%
4015 \global\let\@numberstringspanish\@numberstringspanish

```

As above, but for ordinals

```
4016 \newcommand*{\@@ordinalstringspanish[2]}{%
4017 \@strctr=#1\relax
4018 \ifnum#1>99999
4019 \PackageError{fmtcount}{Out of range}%
4020 {This macro only works for values less than 100000}%
4021 \else
4022 \ifnum#1<0
4023 \PackageError{fmtcount}{Negative numbers not permitted}%
4024 {This macro does not work for negative numbers, however
4025 you can try typing "minus" first, and then pass the modulus of
4026 this number}%
4027 \else
4028 \def#2{}%
4029 \ifnum\@strctr>999\relax
4030 \divide\@strctr by 1000\relax
4031 \ifnum\@strctr>1\relax
4032 \ifnum\@strctr>9\relax
4033 \@tmpstrctr=\@strctr
4034 \ifnum\@strctr<20
4035 \@FCmodulo{\@tmpstrctr}{10}%
4036 \let\@@fc@ordstr#2\relax
4037 \edef#2{\@@fc@ordstr\@teenthstring{\@tmpstrctr}}%
4038 \else
4039 \divide\@tmpstrctr by 10\relax
4040 \let\@@fc@ordstr#2\relax
4041 \edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
4042 \@tmpstrctr=\@strctr
4043 \@FCmodulo{\@tmpstrctr}{10}%
4044 \ifnum\@tmpstrctr>0\relax
4045 \let\@@fc@ordstr#2\relax
4046 \edef#2{\@@fc@ordstr\@unitthstring{\@tmpstrctr}}%
4047 \fi
4048 \fi
4049 \else
4050 \let\@@fc@ordstr#2\relax
4051 \edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
4052 \fi
4053 \fi
4054 \let\@@fc@ordstr#2\relax
4055 \edef#2{\@@fc@ordstr\@thousandth}%
4056 \fi
4057 \@strctr=#1\relax
4058 \@FCmodulo{\@strctr}{1000}%
4059 \ifnum\@strctr>99\relax
4060 \@tmpstrctr=\@strctr
4061 \divide\@tmpstrctr by 100\relax
4062 \ifnum#1>1000\relax
4063 \let\@@fc@ordstr#2\relax
```

```

4064 \edef#2{\@fc@ordstr\ }%
4065 \fi
4066 \let\@fc@ordstr#2\relax
4067 \edef#2{\@fc@ordstr\@hundredthstring{\@tmpstrctr}}%
4068 \fi
4069 \@FCmodulo{\@strctr}{100}%
4070 \ifnum#1>99\relax
4071 \ifnum\@strctr>0\relax
4072 \let\@fc@ordstr#2\relax
4073 \edef#2{\@fc@ordstr\ }%
4074 \fi
4075 \fi
4076 \ifnum\@strctr>19\relax
4077 \@tmpstrctr=\@strctr
4078 \divide\@tmpstrctr by 10\relax
4079 \let\@fc@ordstr#2\relax
4080 \edef#2{\@fc@ordstr\@tenthstring{\@tmpstrctr}}%
4081 \@tmpstrctr=\@strctr
4082 \@FCmodulo{\@tmpstrctr}{10}%
4083 \ifnum\@tmpstrctr>0\relax
4084 \let\@fc@ordstr#2\relax
4085 \edef#2{\@fc@ordstr\ \@unitthstring{\@tmpstrctr}}%
4086 \fi
4087 \else
4088 \ifnum\@strctr>9\relax
4089 \@FCmodulo{\@strctr}{10}%
4090 \let\@fc@ordstr#2\relax
4091 \edef#2{\@fc@ordstr\@teenthstring{\@strctr}}%
4092 \else
4093 \ifnum\@strctr=0\relax
4094 \ifnum#1=0\relax
4095 \let\@fc@ordstr#2\relax
4096 \edef#2{\@fc@ordstr\@unitstring{0}}%
4097 \fi
4098 \else
4099 \let\@fc@ordstr#2\relax
4100 \edef#2{\@fc@ordstr\@unitthstring{\@strctr}}%
4101 \fi
4102 \fi
4103 \fi
4104 \fi
4105 \fi
4106 }%
4107 \global\let\@ordinalstringspanish\@ordinalstringspanish

```

10.1.17 fc-UKenglish.def

English definitions

```
4108 \ProvidesFCLanguage{UKenglish}[2013/08/17]%

```

Loaded fc-english.def if not already loaded

```
4109 \FCloadlang{english}%
```

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

```
4110 \global\let\@ordinalMUKenglish\@ordinalMenglish
4111 \global\let\@ordinalFUKenglish\@ordinalMenglish
4112 \global\let\@ordinalNUKenglish\@ordinalMenglish
4113 \global\let\@numberstringMUKenglish\@numberstringMenglish
4114 \global\let\@numberstringFUKenglish\@numberstringMenglish
4115 \global\let\@numberstringNUKenglish\@numberstringMenglish
4116 \global\let\@NumberstringMUKenglish\@NumberstringMenglish
4117 \global\let\@NumberstringFUKenglish\@NumberstringMenglish
4118 \global\let\@NumberstringNUKenglish\@NumberstringMenglish
4119 \global\let\@ordinalstringMUKenglish\@ordinalstringMenglish
4120 \global\let\@ordinalstringFUKenglish\@ordinalstringMenglish
4121 \global\let\@ordinalstringNUKenglish\@ordinalstringMenglish
4122 \global\let\@OrdinalstringMUKenglish\@OrdinalstringMenglish
4123 \global\let\@OrdinalstringFUKenglish\@OrdinalstringMenglish
4124 \global\let\@OrdinalstringNUKenglish\@OrdinalstringMenglish
```

10.1.18 fc-USenglish.def

US English definitions

```
4125 \ProvidesFCLanguage{USenglish}[2013/08/17]%
```

Loaded fc-english.def if not already loaded

```
4126 \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.)

```
4127 \global\let\@ordinalMUSenglish\@ordinalMenglish
4128 \global\let\@ordinalFUSenglish\@ordinalMenglish
4129 \global\let\@ordinalNUSenglish\@ordinalMenglish
4130 \global\let\@numberstringMUSenglish\@numberstringMenglish
4131 \global\let\@numberstringFUSenglish\@numberstringMenglish
4132 \global\let\@numberstringNUSenglish\@numberstringMenglish
4133 \global\let\@NumberstringMUSenglish\@NumberstringMenglish
4134 \global\let\@NumberstringFUSenglish\@NumberstringMenglish
4135 \global\let\@NumberstringNUSenglish\@NumberstringMenglish
4136 \global\let\@ordinalstringMUSenglish\@ordinalstringMenglish
4137 \global\let\@ordinalstringFUSenglish\@ordinalstringMenglish
4138 \global\let\@ordinalstringNUSenglish\@ordinalstringMenglish
4139 \global\let\@OrdinalstringMUSenglish\@OrdinalstringMenglish
4140 \global\let\@OrdinalstringFUSenglish\@OrdinalstringMenglish
4141 \global\let\@OrdinalstringNUSenglish\@OrdinalstringMenglish
```

10.2 fcnumparser.sty

```
4142 \NeedsTeXFormat{LaTeX2e}
```

```
4143 \ProvidesPackage{fcnumparser}[2017/06/15]
```

```

\fc@counter@parser is just a shorthand to parse a number held in a counter.
4144 \def\fc@counter@parser#1{%
4145   \expandafter\fc@number@parser\expandafter{\the#1.}%
4146 }
4147 \newcount\fc@digit@counter
4148
4149 \def\fc@end@{\fc@end}

```

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.

```

4150 \def\fc@number@analysis#1\fc@nil{%
  First check for the presence of a decimal point in the number.
4151   \def\@tempb##1.##2\fc@nil{\def\fc@integer@part{##1}\def\@tempa{##2}}%
4152   \@tempb#1.\fc@end\fc@nil
4153   \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.

```

4154   \def\@tempb##1,##2\fc@nil{\def\fc@integer@part{##1}\def\@tempa{##2}}%
4155   \@tempb#1,\fc@end\fc@nil
4156   \ifx\@tempa\fc@end@

```

No comma either, so fractional part is set empty.

```

4157   \def\fc@fractional@part{}%
4158   \else

```

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

```

4159   \def\@tempb##1,\fc@end{\def\fc@fractional@part{##1}}%
4160   \expandafter\@tempb\@tempa
4161   \fi
4162   \else

```

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

```

4163   \def\@tempb##1.\fc@end{\def\fc@fractional@part{##1}}%
4164   \expandafter\@tempb\@tempa
4165   \fi
4166 }

```

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>.

```

4167 \def\fc@number@parser#1{%
  First remove all the spaces in #1, and place the result into \@tempa.
4168   \let\@tempa@empty
4169   \def\@tempb##1##2\fc@nil{%
4170     \def\@tempc{##1}%
4171     \ifx\@tempc\space

```

```

4172   \else
4173     \expandafter\def\expandafter\@tempa\expandafter{\@tempa ##1}%
4174   \fi
4175   \def\@tempc{##2}%
4176   \ifx\@tempc\@empty
4177     \expandafter\@gobble
4178   \else
4179     \expandafter\@tempb
4180   \fi
4181   ##2\fc@nil
4182 }%
4183 \@tempb#1\fc@nil

  Get the sign into \fc@sign and the unsigned number part into \fc@number.
4184 \def\@tempb##1##2\fc@nil{\def\fc@sign{##1}\def\fc@number{##2}}%
4185 \expandafter\@tempb\@tempa\fc@nil
4186 \expandafter\if\fc@sign+%
4187   \def\fc@sign@case{1}%
4188 \else
4189   \expandafter\if\fc@sign-%
4190     \def\fc@sign@case{2}%
4191   \else
4192     \def\fc@sign{}%
4193     \def\fc@sign@case{0}%
4194     \let\fc@number\@tempa
4195   \fi
4196 \fi
4197 \ifx\fc@number\@empty
4198   \PackageError{fcnumparser}{Invalid number}{Number must contain at least one non blank
4199     character after sign}%
4200 \fi

  Now, split \fc@number into \fc@integer@part and \fc@fractional@part.
4201 \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 ×
  \fc@unit@weight – \fc@max@weight upto \fc@unit@weight – 1.
4202 \expandafter\fc@digit@counter\fc@unit@weight
4203 \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:

      \fc@max@weight ← (–\fc@digit@counter) + 2 × \fc@unit@weight – 1
4204 \fc@digit@counter – \fc@digit@counter
4205 \advance\fc@digit@counter by \fc@unit@weight
4206 \advance\fc@digit@counter by \fc@unit@weight
4207 \advance\fc@digit@counter by -1 %

```

4208 \edef\fc@max@weight{\the\fc@digit@counter}%
 Now we loop for $i = \text{\fc@unit@weight}$ to \fc@max@weight in order to copy all the digits from $\text{\fc@digit@}(i + \text{offset})$ to $\text{\fc@digit@}(i)$. First we compute offset into \@temp i .

```
4209 {%
4210   \count0 \fc@unit@weight\relax
4211   \count1 \fc@max@weight\relax
4212   \advance\count0 by -\count1 %
4213   \advance\count0 by -1 %
4214   \def\@tempa##1{\def\@tempb{\def\@temp i{##1}}}%
4215   \expandafter\@tempa\expandafter{\the\count0}%
4216   \expandafter
4217 }\@tempb
```

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

```
4218 \expandafter\fc@digit@counter\fc@unit@weight
4219 \def\@temp l{%
4220   \ifnum\fc@digit@counter>\fc@max@weight
4221     \let\next\relax
4222   \else
```

Here is the loop body:

```
4223   {%
4224     \count0 \@temp i
4225     \advance\count0 by \fc@digit@counter
4226     \expandafter\def\expandafter\@temp d\expandafter{\csname fc@digit@the\count0\endcsname
4227     \expandafter\def\expandafter\@temp e\expandafter{\csname fc@digit@the\fc@digit@counte
4228     \def\@tempa###1###2{\def\@tempb{\let###1###2}}%
4229     \expandafter\expandafter\expandafter\@tempa\expandafter\@temp e\@temp d
4230     \expandafter
4231     }\@tempb
4232     \advance\fc@digit@counter by 1 %
4233     \fi
4234     \next
4235   }%
4236   \let\next\@temp l
4237 \@temp l
```

Split $\text{\fc@fractional@part}$ into a sequence of $\text{\fc@digit@}(n)$ 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.

```
4238 \expandafter\fc@digit@counter\fc@unit@weight
4239 \expandafter\fc@parse@integer@digits\fc@fractional@part\fc@end\fc@nil
4240 \edef\fc@min@weight{\the\fc@digit@counter}%
4241 }
```

Macro $\text{\fc@parse@integer@digits}$ is used to

```
4242 \ifcsundef\fc@parse@integer@digits\{}{%
4243   \PackageError{fcnumparser}{Duplicate definition}{Redefinition of
4244   macro ‘\fc@parse@integer@digits’}}
4245 \def\fc@parse@integer@digits#1#2\fc@nil{%
4246   \def\@tempa{#1}%
```

```

4247 \ifx\@tempa\fc@end@
4248 \def\next##1\fc@nil{}%
4249 \else
4250 \let\next\fc@parse@integer@digits
4251 \advance\fc@digit@counter by -1
4252 \expandafter\def\csname fc@digit@\the\fc@digit@counter\endcsname{#1}%
4253 \fi
4254 \next#2\fc@nil
4255 }
4256
4257
4258 \newcommand*{\fc@unit@weight}{0}
4259

```

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

`\fc@read@unit` `\fc@read@unit` just reads one digit and form an integer in the range [0..9]. First we check that the macro is not yet defined.

```

4260 \ifcsundef{fc@read@unit}{-}{%
4261 \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro 'fc@read@unit'}}

```

Arguments as follows:

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

```

4262 \def\fc@read@unit#1#2{%
4263 \ifnum#2>\fc@max@weight
4264 #1=0\relax
4265 \else
4266 \ifnum#2<\fc@min@weight
4267 #1=0\relax
4268 \else
4269 {%
4270 \edef\@tempa{\number#2}%
4271 \count0=\@tempa
4272 \edef\@tempa{\csname fc@digit@\the\count0\endcsname}%
4273 \def\@tempb##1{\def\@tempa{#1=##1\relax}}%
4274 \expandafter\@tempb\expandafter{\@tempa}%
4275 \expandafter
4276 }\@tempa
4277 \fi
4278 \fi
4279 }

```

`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.

```

4280 \ifcsundef{fc@read@hundred}{-}{%
4281 \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

```

4282 \def\fc@read@hundred#1#2{%
4283   {%
4284     \fc@read@unit{\count0}{#2}%
4285     \def\@tempa##1{\fc@read@unit{\count1}{##1}}%
4286     \count2=#2%
4287     \advance\count2 by 1 %
4288     \expandafter\@tempa{\the\count2}%
4289     \multiply\count1 by 10 %
4290     \advance\count1 by \count0 %
4291     \def\@tempa##1{\def\@tempb{#1=##1\relax}}
4292     \expandafter\@tempa\expandafter{\the\count1}%
4293     \expandafter
4294   }\@tempb
4295 }

```

`\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.

```

4296 \ifcsundef{fc@read@thousand}{%
4297   \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro
4298     '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

```

4299 \def\fc@read@thousand#1#2{%
4300   {%
4301     \fc@read@unit{\count0}{#2}%
4302     \def\@tempa##1{\fc@read@hundred{\count1}{##1}}%
4303     \count2=#2%
4304     \advance\count2 by 1 %
4305     \expandafter\@tempa{\the\count2}%
4306     \multiply\count1 by 10 %
4307     \advance\count1 by \count0 %
4308     \def\@tempa##1{\def\@tempb{#1=##1\relax}}
4309     \expandafter\@tempa\expandafter{\the\count1}%
4310     \expandafter
4311   }\@tempb
4312 }

```

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

```

4313 \ifcsundef{fc@read@myriad}{%
4314   \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro
4315     '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

```

4316 \def\fc@read@myriad#1#2{%
4317   {%
4318     \fc@read@hundred{\count0}{#2}%
4319     \def\@tempa##1{\fc@read@hundred{\count1}{##1}}%
4320     \count2=#2
4321     \advance\count2 by 2
4322     \expandafter\@tempa{\the\count2}%
4323     \multiply\count1 by 100 %
4324     \advance\count1 by \count0 %
4325     \def\@tempa##1{\def\@tempb{#1=##1\relax}}%
4326     \expandafter\@tempa\expandafter{\the\count1}%
4327     \expandafter
4328   }\@tempb
4329 }

```

`\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.

```

4330 \ifcsundef{fc@check@nonzeros}{}%
4331 \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro
4332   '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.

```

4333 \def\fc@check@nonzeros#1#2#3{%
4334   {%

```

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

```

4335     \edef\@tempa{\number#1}%
4336     \edef\@tempb{\number#2}%
4337     \count0=\@tempa
4338     \count1=\@tempb\relax

```

Then we do the real job

```

4339     \fc@@check@nonzeros@inner

```

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

```

4340     \def\@tempd##1{\def\@tempa{\def#3{##1}}}%
4341     \expandafter\@tempd\expandafter{\@tempc}%
4342     \expandafter
4343   }\@tempa
4344 }

```

`\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.
4345 \def\fc@@check@nonzeros@inner{%
4346   \ifnum\count0<\fc@min@weight
4347     \count0=\fc@min@weight\relax
4348   \fi
4349   \ifnum\count1>\fc@max@weight\relax
4350     \count1=\fc@max@weight
4351   \fi
4352   \count2\count0 %
4353   \advance\count2 by 1 %
4354   \ifnum\count0>\count1 %
4355     \PackageError{fcnumparser}{Unexpected arguments}{Number in argument 2 of macro
4356       'fc@check@nonzeros' must be at least equal to number in argument 1}%
4357   \else
4358     \fc@@check@nonzeros@inner@loopbody
4359     \ifnum\@tempc>0 %
4360       \ifnum\@tempc<9 %
4361         \ifnum\count0>\count1 %
4362           \else
4363             \let\@tempd\@tempc
4364             \fc@@check@nonzeros@inner@loopbody
4365             \ifnum\@tempc=0 %
4366               \let\@tempc\@tempd
4367             \else
4368               \def\@tempc{9}%
4369             \fi
4370           \fi
4371         \fi
4372       \fi
4373     \fi
4374 }
4375 \def\fc@@check@nonzeros@inner@loopbody{%
4376   % \@tempc <- digit of weight \count0
4377   \expandafter\let\expandafter\@tempc\csname fc@digit@the\count0\endcsname
4378   \advance\count0 by 1 %
4379   \ifnum\@tempc=0 %
4380     \ifnum\count0>\count1 %
4381       \let\next\relax
4382     \else
4383       \let\next\fc@@check@nonzeros@inner@loopbody

```

```

4384     \fi
4385   \else
4386     \ifnum\count0>\count2 %
4387       \def\@tempc{9}%
4388     \fi
4389     \let\next\relax
4390   \fi
4391   \next
4392 }

```

`\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.

```

4393 \ifcsundef{fc@intpart@find@last}{-}{-}%
4394 \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro
4395   '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.

```

4396 \def\fc@intpart@find@last#1{%
4397   {%

```

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

```

4398   \count0=\fc@unit@weight\space
4399   \ifnum\count0<\fc@min@weight\space
4400     \count0=\fc@min@weight\space
4401   \fi

```

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

```

4402   \def\@templ{%
4403     \ifnum\csname fc@digit@the\count0\endcsname=0 %
4404       \advance\count0 by 1 %
4405       \ifnum\count0>\fc@max@weight\space
4406         \let\next\relax
4407       \fi
4408     \else
4409       \let\next\relax
4410     \fi
4411     \next
4412   }%
4413   \let\next\@templ
4414   \@templ

```

Now propagate result after closing bracket into counter #1.

```

4415     \toks0{#1}%
4416     \edef\@tempa{\the\toks0=\the\count0}%
4417     \expandafter
4418   }\@tempa\space
4419 }

```

```

c@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
4420 \ifcsundef{fc@get@last@word}{\PackageError{fcnumparser}{Duplicate definition}{Redefinition
4421 of macro 'fc@get@last@word'}}%
4422 \def\fc@get@last@word#1#2#3{%
4423 {%

First we split #1 into two parts: everything that is upto \fc@wcase exclusive goes to \toks0,
and evrything from \fc@wcase exclusive upto the final \@nil exclusive goes to \toks1.
4424 \def\@tempa##1\fc@wcase##2\@nil\fc@end{%
4425 \toks0{##1}%

Actually a dummy \fc@wcase is appended to \toks1, because that makes easier further
checking that it does not contains any other \fc@wcase.
4426 \toks1{##2\fc@wcase}%
4427 }%
4428 \@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@wcase inside \toks1 other than the tailing dummy one. To that purpose we will
loop while we find that \toks1 contains some \fc@wcase. First we define \@tempa to split
\the\toks1 between parts before and after some potential \fc@wcase.
4429 \def\@tempa##1\fc@wcase##2\fc@end{%
4430 \toks2{##1}%
4431 \def\@tempb{##2}%
4432 \toks3{##2}%
4433 }%

\@tempt is just an aliases of \toks0 to make its handling easier later on.
4434 \toksdef\@tempt0 %

Now the loop itself, this is done by terminal recursion with macro \@templ.
4435 \def\@templ{%
4436 \expandafter\@tempa\the\toks1 \fc@end
4437 \ifx\@tempb\@empty

\@tempb empty means that the only \fc@wcase found in \the\toks1 is the dummy one. So
we end the loop here, \toks2 contains the last word.
4438 \let\next\relax
4439 \else

\@tempb is not empty, first we use
4440 \expandafter\expandafter\expandafter\@tempt
4441 \expandafter\expandafter\expandafter{%
4442 \expandafter\the\expandafter\@tempt
4443 \expandafter\fc@wcase\the\toks2}%
4444 \toks1\toks3 %
4445 \fi
4446 \next
4447 }%

```

```

4448   \let\next\@templ
4449   \@templ
4450   \edef\@tempa{\def\noexpand#2{\the\toks0}\def\noexpand#3{\the\toks2}}%
4451   \expandafter
4452   }\@tempa
4453 }

```

c@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

```

4454 \ifcsundef{fc@get@last@letter}{-}{\PackageError{fcnumparser}{Duplicate definition}{Redefinition
4455   of macro 'fc@get@last@letter'}}%
4456 \def\fc@get@last@letter#1#2#3{%
4457   {%

```

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.

```

4458   \toks1{#1}%
4459   \toks0{}%
4460   \toksdef\@tempt0 %

```

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

```

4461   \def\@tempa##1##2\fc@nil{%
4462     \toks2{##1}%
4463     \toks3{##2}%
4464     \def\@tempb{##2}%
4465   }%

```

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

```

4466   \def\@templ{%
4467     \expandafter\@tempa\the\toks1 \fc@nil
4468     \ifx\@tempb\@empty

```

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

```

4469     \let\next\relax
4470   \else

```

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

```

4471     \expandafter\expandafter\expandafter\@tempt
4472     \expandafter\expandafter\expandafter{%
4473     \expandafter\the\expandafter\@tempt
4474     \the\toks2}%

```

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

```

4475     \toks1\toks3 %
4476     \fi
4477     \next
4478   }%

```

Here run the loop.

```

4479   \let\next\@templ
4480   \next

```

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

```
4481 \edef\@tempa{\def\noexpand#2{\the\toks0}\def\noexpand#3{\the\toks1}}%
4482 \expandafter
4483 }\@tempa
4484 }%
```

10.3 fcprefix.sty

Pseudo-latin prefixes.

```
4485 \NeedsTeXFormat{LaTeX2e}
4486 \ProvidesPackage{fcprefix}[2012/09/28]
4487 \RequirePackage{ifthen}
4488 \RequirePackage{keyval}
4489 \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’.

```
4490 \define@key{fcprefix}{use duode and unde}[below20]{%
4491 \ifthenelse{\equal{#1}{below20}}{%
4492 \def\fc@duodeandunde{2}%
4493 }{%
4494 \ifthenelse{\equal{#1}{never}}{%
4495 \def\fc@duodeandunde{0}%
4496 }{%
4497 \PackageError{fcprefix}{Unexpected option}{%
4498 Option ‘use duode and unde’ expects ‘below 20’ or ‘never’ }%
4499 }%
4500 }%
4501 }
```

Default is ‘below 20’ like in French.

```
4502 \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.

```
4503 \define@key{fcprefix}{numeral u in duo}[false]{%
4504 \ifthenelse{\equal{#1}{false}}{%
4505 \let\fc@u@in@duo\@empty
4506 }{%
4507 \ifthenelse{\equal{#1}{true}}{%
4508 \def\fc@u@in@duo{u}%
4509 }{%
4510 \PackageError{fcprefix}{Unexpected option}{%
4511 Option ‘numeral u in duo’ expects ‘true’ or ‘false’ }%
4512 }%
4513 }%
4514 }
```

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.

```

4515 \define@key{fcprefix}{e accute}[false]{%
4516   \ifthenelse{\equal{#1}{false}}{%
4517     \let\fc@prefix@eaccute\@firstofone
4518   }{%
4519     \ifthenelse{\equal{#1}{true}}{%
4520       \let\fc@prefix@eaccute\'%
4521     }{%
4522       \PackageError{fcprefix}{Unexpected option}{%
4523         Option 'e accute' expects 'true' or 'false' }%
4524     }%
4525   }%
4526 }

```

Default is to set accute accent like in French.

```
4527 \let\fc@prefix@eaccute\'%
```

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

recursive for which millia squared is noted as 'milliamillia'

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

prefix for which millia squared is noted as 'bismillia'

```

4528 \define@key{fcprefix}{power of millia}[prefix]{%
4529   \ifthenelse{\equal{#1}{prefix}}{%
4530     \let\fc@power@of@millia@init\@gobbletwo
4531     \let\fc@power@of@millia\fc@prefix@millia
4532   }{%
4533     \ifthenelse{\equal{#1}{arabic}}{%
4534       \let\fc@power@of@millia@init\@gobbletwo
4535       \let\fc@power@of@millia\fc@@arabic@millia
4536     }{%
4537       \ifthenelse{\equal{#1}{recursive}}{%
4538         \let\fc@power@of@millia@init\fc@@recurse@millia@init
4539         \let\fc@power@of@millia\fc@@recurse@millia
4540       }{%
4541         \PackageError{fcprefix}{Unexpected option}{%
4542           Option 'power of millia' expects 'recursive', 'arabic', or 'prefix' }%
4543       }%
4544     }%
4545   }%
4546 }

```

Arguments as follows:

#1 output macro

#2 number with current weight w

```

4547 \def\fc@@recurse@millia#1#2{%
4548   \let\@temp#1%
4549   \edef#1{millia\@temp}%
4550 }

```

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


```

#1  output macro
#2  number with current weight  $w$ 
4551 \def\fc@@recurse@millia@init#1#2{%
4552  {%
    Save input argument current weight  $w$  into local macro \@tempb.
4553    \edef\@tempb{\number#2}%
    Now main loop from 0 to  $w$ . Final value of \@tempa will be the result.
4554    \count0=0 %
4555    \let\@tempa\@empty
4556    \loop
4557      \ifnum\count0<\@tempb
4558        \advance\count0 by 1 %
4559        \expandafter\def
4560        \expandafter\@tempa\expandafter{\@tempa millia}%
4561    \repeat
    Now propagate the expansion of \@tempa into #1 after closing brace.
4562    \edef\@tempb{\def\noexpand#1{\@tempa}}%
4563    \expandafter
4564  }\@tempb
4565 }

```

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

```

#1  output macro
#2  number with current weight  $w$ 
4566 \def\fc@@arabic@millia#1#2{%
4567  \ifnum#2=0 %
4568    \let#1\@empty
4569  \else
4570    \edef#1{millia\^{\the#2}}%
4571  \fi
4572 }

```

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

```

#1  output macro
#2  number with current weight  $w$ 
4573 \def\fc@@prefix@millia#1#2{%
4574  \fc@@latin@numeral@pefix{#2}{#1}%
4575 }

```

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

```

4576 \let\fc@power@of@millia@init\@gobbletwo
4577 \let\fc@power@of@millia\fc@@prefix@millia

```

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.

```

4578 \ifcsundef\fc@@latin@cardinal@pefix{}{%
4579  \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro ‘fc@@latin@cardinal@pefix

```

Arguments as follows:

#1 input number to be formatted

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

```
4580 \def\fc@@latin@cardinal@pefix#1#2{%
```

```
4581  {%
```

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

```
4582  \edef\@tempa{\number#1}%
```

Now parse number from expanded input.

```
4583  \expandafter\fc@number@parser\expandafter{\@tempa}%
```

```
4584  \count2=0 %
```

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

```
4585  \let\@tempt\@empty
```

```
4586  \def\@tempu{t}%
```

\@tempm will hold the milliaⁿ⁺³

```
4587  \let\@tempm\@empty
```

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

```
4588  \def\@templ{%
```

```
4589    \ifnum\count2>\fc@max@weight
```

```
4590      \let\next\relax
```

```
4591    \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.

```
4592      \fc@read@unit{\count3}{\count2}%
```

```
4593      \advance\count2 by 1 %
```

```
4594      \fc@read@unit{\count4}{\count2}%
```

```
4595      \advance\count2 by 1 %
```

```
4596      \fc@read@unit{\count5}{\count2}%
```

```
4597      \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.

```
4598      \edef\@tempn{%
```

```
4599        \ifnum\count3=0\else 1\fi
```

```
4600        \ifnum\count4=0\else 1\fi
```

```
4601        \ifnum\count5=0\else 1\fi
```

```
4602      }%
```

```
4603      \ifx\@tempn\@empty\else
```

```
4604        \let\@tempt\@tempu
```

```
4605        \let\@tempu\@empty
```

```
4606      \fi
```

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

```
4607      \let\@temp\@tempa
```

```
4608      \edef\@tempa{%
```

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

```

4609         \ifcase\count5 %
4610         \or cen%
4611         \or ducen%
4612         \or trecen%
4613         \or quadringen%
4614         \or quingen%
4615         \or sescen%
4616         \or septigen%
4617         \or octingen%
4618         \or nongen%
4619         \fi

```

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

```

4620         \ifnum\count4=0 %
4621         % x0(0..9)
4622         \ifnum\count2=3 %
4623         % Absolute weight zero
4624         \ifcase\count3 \@tempt
4625         \or m%
4626         \or b%
4627         \or tr%
4628         \or quadr%
4629         \or quin\@tempt
4630         \or sex\@tempt
4631         \or sep\@tempt
4632         \or oc\@tempt
4633         \or non%
4634         \fi
4635         \else

```

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

```

4636         \ifcase\count3 %
4637         \or \ifnum\count2>\fc@max@weight\else un\fi
4638         \or d\fc@u@in@duo o%
4639         \or tre%
4640         \or quattuor%
4641         \or quin%
4642         \or sex%
4643         \or septen%
4644         \or octo%
4645         \or novem%
4646         \fi
4647         \fi
4648         \else
4649         % x(10..99)
4650         \ifcase\count3 %
4651         \or un%
4652         \or d\fc@u@in@duo o%
4653         \or tre%

```

```

4654         \or quattuor%
4655         \or quin%
4656         \or sex%
4657         \or septen%
4658         \or octo%
4659         \or novem%
4660         \fi
4661         \ifcase\count4 %
4662         \or dec%
4663         \or vigin\@tempt
4664         \or trigin\@tempt
4665         \or quadragin\@tempt
4666         \or quinquagin\@tempt
4667         \or sexagin\@tempt
4668         \or septuagin\@tempt
4669         \or octogin\@tempt
4670         \or nonagin\@tempt
4671         \fi
4672     \fi

```

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

```
4673     \@tempm
```

And append previous version of \@tempa.

```
4674     \@tempm
4675     }%
```

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

```

4676     \fc@power@of@millia\@tempm{\count2}%
4677     \fi
4678     \next
4679     }%
4680     \let\@tempa\@empty
4681     \let\next\@templ
4682     \@templ

```

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

```

4683     \def\@tempb##1{\def\@tempa{\def#2{##1}}}%
4684     \expandafter\@tempb\expandafter{\@tempa}%
4685     \expandafter
4686     }\@tempa
4687 }

```

For 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.

```

4688 \ifcsundef{fc@latin@numeral@pefix}{-}{%
4689 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro

```

```
4690   'fc@@latin@numeral@pefix'}}}
```

Arguments as follows:

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

```
4691 \def\fc@@latin@numeral@pefix#1#2{%
4692   {%
4693     \edef\@tempa{\number#1}%
4694     \def\fc@unit@weight{0}%
4695     \expandafter\fc@number@parser\expandafter{\@tempa}%
4696     \count2=0 %
```

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

```
4697   \let\@tempm\@empty
```

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

```
4698   \def\@templ{%
4699     \ifnum\count2>\fc@max@weight
4700       \let\next\relax
4701     \else
```

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

```
4702       \fc@read@unit{\count3}{\count2}%
4703       \advance\count2 by 1 %
4704       \fc@read@unit{\count4}{\count2}%
4705       \advance\count2 by 1 %
4706       \fc@read@unit{\count5}{\count2}%
4707       \advance\count2 by 1 %
```

Check the use of duodevicies instead of octodecies.

```
4708       \let\@tempn\@secondoftwo
4709       \ifnum\count3>7 %
4710         \ifnum\count4<\fc@duodeandunde
4711           \ifnum\count4>0 %
4712             \let\@tempn\@firstoftwo
4713           \fi
4714         \fi
4715       \fi
4716       \@tempn
4717       {% use duodevicies for eighteen
4718         \advance\count4 by 1 %
4719         \let\@temps\@secondoftwo
4720       }{% do not use duodevicies for eighteen
4721         \let\@temps\@firstoftwo
4722       }%
4723       \let\@temp\@tempa
4724       \edef\@tempa{%
4725         % hundreds
4726         \ifcase\count5 %
4727         \expandafter\@gobble
```

```

4728         \or c%
4729         \or duc%
4730         \or trec%
4731         \or quadring%
4732         \or quing%
4733         \or sesc%
4734         \or septing%
4735         \or octing%
4736         \or nong%
4737         \fi
4738         {enties}%
4739         \ifnum\count4=0 %

```

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

```

4740         \ifcase\count3 %
4741         \or
4742         \ifnum\count2=3 %
4743             s\fc@prefix@eacute emel%
4744         \else
4745             \ifnum\count2>\fc@max@weight\else un\fi
4746         \fi
4747         \or bis%
4748         \or ter%
4749         \or quater%
4750         \or quinquies%
4751         \or sexies%
4752         \or septies%
4753         \or octies%
4754         \or novies%
4755         \fi
4756         \else

```

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

```

4757         \ifcase\count3 %
4758         \or un%
4759         \or d\fc@u@in@duo o%
4760         \or ter%
4761         \or quater%
4762         \or quin%
4763         \or sex%
4764         \or septen%
4765         \or \@temps{octo}{duod\fc@prefix@eacute e}% x8 = two before next (x+1)0
4766         \or \@temps{novem}{und\fc@prefix@eacute e}% x9 = one before next (x+1)0
4767         \fi
4768         \ifcase\count4 %
4769         % can't get here
4770         \or d\fc@prefix@eacute ec%
4771         \or vic%
4772         \or tric%
4773         \or quadrag%

```

```

4774         \or quinquag%
4775         \or sexag%
4776         \or septuag%
4777         \or octog%
4778         \or nonag%
4779         \fi
4780         ies%
4781     \fi
4782     % Insert the millies(n/3) only if one of \count3 \count4 \count5 is non zero
4783     \@tempm
4784     % add up previous version of \@tempa
4785     \@tempm
4786 }%

```

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

```

4787     \let\@tempm\@tempm
4788     \edef\@tempm{millies\@tempm}%
4789     \fi
4790     \next
4791 }%
4792 \let\@tempa\@empty
4793 \let\next\@templ
4794 \@templ

```

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

```

4795     \def\@tempb##1{\def\@tempa{\def#2{##1}}}%
4796     \expandafter\@tempb\expandafter{\@tempa}%
4797     \expandafter
4798 } \@tempa
4799 }

```

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.

```

4800 \def\fc@call@opt@arg@second#1#2{%
4801   \def\@tempb{%
4802     \ifx[\@tempa
4803       \def\@tempc[####1]{%
4804         {#1{####1}{#2}}%
4805       }%
4806     \else
4807       \def\@tempc{{#1}{#2}}%
4808     \fi
4809     \@tempc
4810   }%
4811   \futurelet\@tempa
4812   \@tempb
4813 }

4814 \def\fc@call@opt@arg@first#1{%
4815   \def\@tempb{%
4816     \ifx[\@tempa
4817       \def\@tempc[####1]####2{{#1{####1}{####2}}}%
4818     \else
4819       \def\@tempc####1{{#1}{####1}}%
4820     \fi
4821     \@tempc
4822   }%
4823   \futurelet\@tempa
4824   \@tempb
4825 }
4826
4827 \let\fc@call\fc@call@opt@arg@first

```

User API.

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

- #1 local options
- #2 input number

```

4828 \newcommand*{\@latinnumeralstringnum}[2]{%
4829   \setkeys{fcprefix}{#1}%
4830   \fc@latin@numeral@pefix{#2}\@tempa
4831   \@tempa
4832 }

```

Arguments as follows:

- #1 local options
- #2 input counter

```

4833 \newcommand*{\@latinnumeralstring}[2]{%
4834   \setkeys{fcprefix}{#1}%
4835   \expandafter\let\expandafter
4836     \@tempa\expandafter\csname c@#2\endcsname
4837   \expandafter\fc@latin@numeral@pefix\expandafter{\the\@tempa}\@tempa
4838   \@tempa
4839 }

```



```

4840 \newcommand*{\latinnumeralstring}{%
4841   \fc@call\@latinnumeralstring
4842 }

4843 \newcommand*{\latinnumeralstringnum}{%
4844   \fc@call\@latinnumeralstringnum
4845 }

```

10.4 fmtcount.sty

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

```

4846 \NeedsTeXFormat{LaTeX2e}
4847 \ProvidesPackage{fmtcount}[2024/08/31 v3.08]
4848 \RequirePackage{ifthen}

4849 \RequirePackage{xkeyval}
4850 \RequirePackage{etoolbox}
4851 \RequirePackage{fcprefix}

```

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

```
4852 \RequirePackage{amsgen}
```

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

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

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

```
4853 \providecommand*\fc@orddef@ult}[1]{\fc@textsuperscript{#1}}
```

```
c@ord@multiling
```

```
4854 \providecommand*\fc@ord@multiling}[1]{%
4855   \ifcsundef{fc@\language@name @alias@of}{%
```

Not a supported language, just use the default setting:

```
4856   \fc@orddef@ult{#1}}{%
```

```
4857   \expandafter\let\expandafter\@tempa\csname fc@\language@name @alias@of\endcsname
4858   \ifcsundef{fc@ord@\@tempa}{%
```

Not language specific setting, just use the default setting:

```
4859   \fc@orddef@ult{#1}}{%
```

Language with specific setting, use that setting:

```
4860 \csname fc@ord@\@tempa\endcsname{#1}}}
```

```
\padzeroes
```

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

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

```

4861 \newcount\c@padzeroesN
4862 \c@padzeroesN=1\relax
4863 \providecommand*\padzeroes}[1][17]{\c@padzeroesN=#1}

```

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

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

```
4864 \newcount\fc@tmpcatcode
4865 \def\fc@languages{}%
4866 \def\fc@mainlang{}%
4867 \newcommand*\FCloadlang[1]{%
4868   \@FC@iflangloaded{#1}{}%
4869   {%
4870     \fc@tmpcatcode=\catcode'\@ \relax
4871     \catcode '@ 11 \relax
4872     \InputIfFileExists{fc-#1.def}%
4873     {%
4874       \ifdefempty{\fc@languages}%
4875       {%
4876         \gdef\fc@languages{#1}%
4877       }%
4878       {%
4879         \gappto\fc@languages{,#1}%
4880       }%
4881       \gdef\fc@mainlang{#1}%
4882     }%
4883   }%
4884   \catcode '@ \fc@tmpcatcode \relax
4885 }%
4886 }
```

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

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

```
4887 \newcommand*\@FC@iflangloaded[3]{%
4888   \ifcsundef{ver@fc-#1.def}{#3}{#2}%
4889 }
```

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

```
4890 \newcommand*\ProvidesFCLanguage[1]{%
4891   \ProvidesFile{fc-#1.def}%
4892 }
```

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

```
4893 \newif\iffmtcount@language@option
4894 \fmtcount@language@optionfalse
```

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

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

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

```
4895 \newcommand*\fc@supported@language@list{%
4896 english,%
4897 UKenglish,%
4898 brazilian,%
4899 british,%
4900 USenglish,%
4901 american,%
4902 spanish,%
4903 portuges,%
4904 portuguese,%
4905 french,%
4906 frenchb,%
4907 francais,%
4908 german,%
4909 germanb,%
4910 ngerman,%
4911 ngermanb,%
4912 italian,%
4913 dutch}
```

`te@on@languages`

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

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

```
4914 \newcommand*\fc@iterate@on@languages[1]{%
4915 \ifx\fc@supported@language@list\empty
```

That case should never happen!

```
4916 \PackageError{fmtcount}{Macro ‘\protect\fc@iterate@on@languages’ is empty}{You should never
4917 Something is broken within \texttt{fmtcount}, please report the issue on
```

```

4918     \texttt{https://github.com/search?q=fmtcount\&ref=cmdform\&type=Issues}}}%
4919 \else
4920   \let\fc@iterate@on@languages@body#1
4921   \expandafter\@fc@iterate@on@languages\fc@supported@language@list,\@nil,%
4922 \fi
4923 }
4924 \def\@fc@iterate@on@languages#1,{%
4925   {%
4926     \def\@tempa{#1}%
4927     \ifx\@tempa\@nnil
4928       \let\@tempa\@empty
4929     \else
4930       \def\@tempa{%
4931         \fc@iterate@on@languages@body{#1}%
4932         \@fc@iterate@on@languages
4933       }%
4934     \fi
4935     \expandafter
4936   }\@tempa
4937 }%

```

polyglossialdf

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

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

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

```

4938 \newcommand*\@fc@loadifbabelldf[1]{\ifcsundef{ver@#1.ldf}{}\FCloadlang{#1}}
4939 \newcommand*\@fc@loadifbabelorpolyglossialdf[1]{
4940 \ifpackageloaded{polyglossia}{%
4941 \def\@fc@loadifbabelorpolyglossialdf#1{\IfFileExists{gloss-#1.ldf}{\ifcsundef{#1@loaded}}{\F
4942 \@fc@loadifbabelldf{#1}%
4943 }%
4944 }\ifpackageloaded{babel}{%
4945 \let\@fc@loadifbabelorpolyglossialdf\@fc@loadifbabelldf
4946 }{}}

```

Load appropriate language definition files:

```
4947 \fc@iterate@on@languages\@fc@loadifbabelorpolyglossialdf
```

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

```

4948 \def\fc@iterate@on@languages@body#1{%
4949 \expandafter\def\csname fc@#1@alias@of\endcsname{#1}}
4950 \expandafter\@fc@iterate@on@languages\fc@supported@language@list,\@nil,%

```

Now define those languages that are aliases of another language. This is done with: `\@tempa {<alias>}{<language>}`

```
4951 \def\@tempa#1#2{%
4952   \expandafter\def\csname fc@#1@alias@of\endcsname{#2}%
4953 }%
4954 \@tempa{frenchb}{french}
4955 \@tempa{français}{french}
4956 \@tempa{germanb}{german}
4957 \@tempa{ngermanb}{german}
4958 \@tempa{ngerman}{german}
4959 \@tempa{british}{english}
4960 \@tempa{american}{USenglish}
```

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

```
4961 \def\fc@iterate@on@languages@body#1{%
4962   \define@key{fmtcount}{#1}[]{%
4963     \@FC@iflangloaded{#1}%
4964     {%
4965       \setkeys{fc\csname fc@#1@alias@of\endcsname}{##1}%
4966     }{%
4967       \PackageError{fmtcount}%
4968         {Language ‘#1’ not defined}%
4969         {You need to load \ifxetex polyglossia\else babel\fi\space before loading fmtcount}%
4970     }%
4971 }%
4972 \ifthenelse{\equal{\csname fc@#1@alias@of\endcsname}{#1}}{%
4973   \define@key{fc\csname fc@#1@alias@of\endcsname}{fmtord}{%
4974     \ifthenelse{\equal{##1}{raise}\or\equal{##1}{level}}{%
4975       \expandafter\let\expandafter\@tempa\csname fc@set@ord@as@##1\endcsname
4976       \expandafter\@tempa\csname fc@ord@#1\endcsname
4977     }{%
4978       \ifthenelse{\equal{##1}{undefine}}{%
4979         \expandafter\let\csname fc@ord@#1\endcsname\undefined
4980       }{%
4981         \PackageError{fmtcount}%
4982           {Invalid value ‘##1’ to fmtord key}%
4983           {Option ‘fmtord’ can only take the values ‘level’, ‘raise’
4984             or ‘undefine’}%
4985       }%
4986     }%
4987 }{%
```

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

```
4988   \expandafter\let\expandafter\@tempa\csname KV@\csname fc@#1@alias@of\endcsname @fmtord\endc
4989   \expandafter\let\csname KV@#1@fmtord\endcsname\@tempa
4990 }%
4991 }
4992 \expandafter\@fc@iterate@on@languages\fc@supported@language@list,\@nil,%
```

`fmtord` Key to determine how to display the ordinal

```
4993 \def\fc@set@ord@as@level#1{%
4994   \def#1##1{##1}%
4995 }
4996 \def\fc@set@ord@as@raise#1{%
4997   \let#1\fc@textsuperscript
4998 }
4999 \define@key{fmtcount}{fmtord}{%
5000   \ifthenelse{\equal{#1}{level}}
5001     \or\equal{#1}{raise}}%
5002   {%
5003     \csname fc@set@ord@as@#1\endcsname\fc@orddef@ult
5004     \def\fmtcount@fmtord{#1}%
5005   }%
5006   {%
5007     \PackageError{fmtcount}%
5008     {Invalid value ‘#1’ to fmtord key}%
5009     {Option ‘fmtord’ can only take the values ‘level’ or ‘raise’}%
5010   }%
5011 }
```

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

```
5012 \newif\iffmtord@abbrv
5013 \fmtord@abbrvtrue
5014 \define@key{fmtcount}{abbrv}[true]{%
5015   \ifthenelse{\equal{#1}{true}\or\equal{#1}{false}}%
5016     {%
5017       \csname fmtord@abbrv#1\endcsname
5018     }%
5019     {%
5020       \PackageError{fmtcount}%
5021       {Invalid value ‘#1’ to fmtord key}%
5022       {Option ‘abbrv’ can only take the values ‘true’ or
5023         ‘false’}%
5024     }%
5025 }
```

`prefix`

```
5026 \define@key{fmtcount}{prefix}[scale=long]{%
5027   \RequirePackage{fmtprefix}%
5028   \fmtprefixsetoption{#1}%
5029 }
```

`countsetoptions` Define command to set options.

```
5030 \def\fmtcountsetoptions{%
5031   \def\fmtcount@fmtord{}}%
```

```
5032 \setkeys{fmtcount}}}%
```

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

```
5033 \InputIfFileExists{fmtcount.cfg}%
5034 {%
5035 \PackageInfo{fmtcount}{Using configuration file fmtcount.cfg}%
5036 }%
5037 {%
5038 }
```

ption@lang@list

```
5039 \newcommand*{\fmtcount@loaded@by@option@lang@list}{}%
```

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

```
5040 \newcommand*{\fc@declare@language@option[1]}{%
5041 \DeclareOption{#1}{%
5042 \ifx\fmtcount@loaded@by@option@lang@list\@empty
5043 \def\fmtcount@loaded@by@option@lang@list{#1}%
5044 \else
5045 \edef\fmtcount@loaded@by@option@lang@list{\fmtcount@loaded@by@option@lang@list,#1}%
5046 \fi
5047 }}%
5048 \fc@iterate@on@languages\fc@declare@language@option
```

level

```
5049 \DeclareOption{level}{\def\fmtcount@fmtord{level}%
5050 \def\fc@orddef@ult#1{#1}}
```

raise

```
5051 \DeclareOption{raise}{\def\fmtcount@fmtord{raise}%
5052 \def\fc@orddef@ult#1{\fc@textsuperscript{#1}}}
```

Process package options

```
5053 \ProcessOptions\relax
```

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

```
5054 \ifx\fmtcount@loaded@by@option@lang@list\@empty\else
5055 \def\fc@iterate@on@languages@body#1{%
5056 \FC@iflangloaded{#1}{%
5057 \fmtcount@language@optiontrue
5058 \FCloadlang{#1}%
5059 }}
5060 \expandafter\fc@iterate@on@languages\fmtcount@loaded@by@option@lang@list,\@nil,%
5061 \fi
```

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

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

```
5062 \newcount \@DT@modctr
5063 \newcommand*{\@FCmodulo}[2]{%
5064   \@DT@modctr=#1\relax
5065   \divide \@DT@modctr by #2\relax
5066   \multiply \@DT@modctr by #2\relax
5067   \advance #1 by -\@DT@modctr
5068 }
```

The following registers are needed by `|ordinal|` etc

```
5069 \newcount \@ordinalctr
5070 \newcount \@orgargctr
5071 \newcount \@strctr
5072 \newcount \@tmpstrctr
```

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

```
5073 \newif\if@DT@padzeroes
5074 \newcount \@DT@loopN
5075 \newcount \@DT@X
```

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

```
5076 \newrobustcmd*{\@binary}[1]{%
5077   \@DT@padzeroestrue
5078   \@DT@loopN=17\relax
5079   \@strctr=\@DT@loopN
5080   \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by \@ne}%
5081   \@strctr=65536\relax
5082   \@DT@X=#1\relax
5083   \loop
5084     \@DT@modctr=\@DT@X
5085     \divide\@DT@modctr by \@strctr
5086     \ifthenelse{\boolean{@DT@padzeroes}
5087       \and \(\@DT@modctr=0\)}
5088       \and \(\@DT@loopN>\c@padzeroesN\)}%
5089     {\the\@DT@modctr}%
5090     \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
5091     \multiply\@DT@modctr by \@strctr
5092     \advance\@DT@X by -\@DT@modctr
5093     \divide\@strctr by \tw@
5094     \advance\@DT@loopN by \m@ne
5095     \ifnum\@strctr>\@ne
5096     \repeat
5097     \the\@DT@X
```



```

5099 }
5100
5101 \let\binarynum=\@binary

```

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

```

5102 \newrobustcmd*{\@octal}[1]{%
5103   \@DT@X=#1\relax
5104   \ifnum\@DT@X>32768
5105     \PackageError{fmtcount}%
5106     {Value of counter too large for \protect\@octal}
5107     {Maximum value 32768}
5108   \else
5109     \@DT@padzeroes\true
5110     \@DT@loopN=6\relax
5111     \@strctr=\@DT@loopN
5112     \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by \@ne}%
5113     \@strctr=32768\relax
5114     \loop
5115       \@DT@modctr=\@DT@X
5116       \divide\@DT@modctr by \@strctr
5117       \ifthenelse{\boolean{\@DT@padzeroes}
5118         \and \(\@DT@modctr=0\}
5119         \and \(\@DT@loopN>\c@padzeroesN\)}%
5120       {\the\@DT@modctr}%
5121       \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
5122       \multiply\@DT@modctr by \@strctr
5123       \advance\@DT@X by -\@DT@modctr
5124       \divide\@strctr by \@viiipt
5125       \advance\@DT@loopN by \m@ne
5126     \ifnum\@strctr>\@ne
5127     \repeat
5128     \the\@DT@X
5129   \fi
5130 }
5131 \let\octalnum=\@octal

```

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

```

5132 \newcommand*{\@@hexadecimal}[1]{%
5133   \ifcase#1\or1\or2\or3\or4\or5\or
5134   6\or7\or8\or9\or a\or b\or c\or d\or e\or f\fi
5135 }

```

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

```

5136 \newrobustcmd*{\hexadecimalnum}{\@hexadecimalengine\@@hexadecimal}

```

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

```

5137 \newcommand*{\@@Hexadecimal}[1]{%
5138   \ifcase#1\or1\or2\or3\or4\or5\or6\or
5139   7\or8\or9\or A\or B\or C\or D\or E\or F\fi

```

5140 }

`\HEXADecimalnum` Uppercase hexadecimal

```
5141 \newrobustcmd*{\HEXADecimalnum}{\@hexadecimalengine\@Hexadecimal}
5142 \newcommand*\@hexadecimalengine[2]{%
5143   \@DT@padzeroestru
5144   \@DT@loopN=\@vpt
5145   \@strctr=\@DT@loopN
5146   \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by \@ne}%
5147   \@strctr=65536\relax
5148   \@DT@X=#2\relax
5149   \loop
5150     \@DT@modctr=\@DT@X
5151     \divide\@DT@modctr by \@strctr
5152     \ifthenelse{\boolean{\@DT@padzeroes}
5153       \and \!(\@DT@modctr=0)
5154       \and \!(\@DT@loopN>\c@padzeroesN)}
5155     {#1\@DT@modctr}%
5156     \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
5157     \multiply\@DT@modctr by \@strctr
5158     \advance\@DT@X by -\@DT@modctr
5159     \divide\@strctr by 16\relax
5160     \advance\@DT@loopN by \m@ne
5161     \ifnum\@strctr>\@ne
5162     \repeat
5163     #1\@DT@X
5164 }
5165 \def\Hexadecimalnum{%
5166   \PackageWarning{fmtcount}{\string\Hexadecimalnum\space is deprecated, use \string\HEXADecimal
5167     instead. The \string\Hexadecimalnum\space control sequence name is confusing as it can misl
5168     that only the 1st letter is upper-cased.}%
5169   \HEXADecimalnum}
```

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

```
5170 \newrobustcmd*{\@aaalph}{\fc@aaalph\@alph}
5171 \newcommand*\fc@aaalph[2]{%
5172   \@DT@loopN=#2\relax
5173   \@DT@X\@DT@loopN
5174   \advance\@DT@loopN by \m@ne
5175   \divide\@DT@loopN by 26\relax
5176   \@DT@modctr=\@DT@loopN
5177   \multiply\@DT@modctr by 26\relax
5178   \advance\@DT@X by \m@ne
5179   \advance\@DT@X by -\@DT@modctr
5180   \advance\@DT@loopN by \@ne
5181   \advance\@DT@X by \@ne
5182   \edef\@tempa{#1\@DT@X}%
5183   \loop
5184     \@tempa
```

```

5185 \advance\@DT@loopN by \m@ne
5186 \ifnum\@DT@loopN>0
5187 \repeat
5188 }
5189
5190 \let\aaalphnum=\@aaalph

```

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

```

5191 \newrobustcmd*{\@AAAAlph}{\fc@aaalph\@Alph}%
5192
5193 \let\AAAalphnum=\@AAAAlph

```

\abalphnum Lowercase alphabetical representation

```

5194 \newrobustcmd*{\@abalph}{\fc@abalph\@alph}%
5195 \newcommand*\fc@abalph[2]{%
5196 \@DT@X=#2\relax
5197 \ifnum\@DT@X>17576\relax
5198 \ifx#1\@alph\def\@tempa{\@abalph}%
5199 \else\def\@tempa{\@ABAlph}\fi
5200 \PackageError{fmtcount}%
5201 {Value of counter too large for \expandafter\protect\@tempa}%
5202 {Maximum value 17576}%
5203 \else
5204 \@DT@padzeroestrue
5205 \@strctr=17576\relax
5206 \advance\@DT@X by \m@ne
5207 \loop
5208 \@DT@modctr=\@DT@X
5209 \divide\@DT@modctr by \@strctr
5210 \ifthenelse{\boolean{\@DT@padzeroes}
5211 \and \(\@DT@modctr=1\)}%
5212 {}{\#1\@DT@modctr}%
5213 \ifnum\@DT@modctr=\@ne\else\@DT@padzeroesfalse\fi
5214 \multiply\@DT@modctr by \@strctr
5215 \advance\@DT@X by -\@DT@modctr
5216 \divide\@strctr by 26\relax
5217 \ifnum\@strctr>\@ne
5218 \repeat
5219 \advance\@DT@X by \@ne
5220 #1\@DT@X
5221 \fi
5222 }
5223
5224 \let\abalphnum=\@abalph

```

\ABAlphnum Uppercase alphabetical representation

```

5225 \newrobustcmd*{\@ABAlph}{\fc@abalph\@Alph}%
5226 \let\ABAlphnum=\@ABAlph

```

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

```
5227 \def\@fmtc@count#1#2\relax{%
5228   \if\relax#1%
5229   \else
5230     \advance\@strctr by 1\relax
5231     \@fmtc@count#2\relax
5232   \fi
5233 }
```

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

```
5234 \newrobustcmd*{\@decimal}[1]{%
5235   \@strctr=0\relax
5236   \expandafter\@fmtc@count\number#1\relax
5237   \@DT@loopN=\c@padzeroesN
5238   \advance\@DT@loopN by -\@strctr
5239   \ifnum\@DT@loopN>0\relax
5240     \@strctr=0\relax
5241     \whiledo{\@strctr < \@DT@loopN}{0\advance\@strctr by 1\relax}%
5242   \fi
5243   \number#1\relax
5244 }
5245
5246 \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.

```
5247 \newcommand{\FCordinal}[1]{%
5248   \ordinalnum{%
5249     \the\value{#1}}%
5250 }
```

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

```
5251 \ifcsundef{ordinal}
5252 {\let\ordinal\FCordinal}%
5253 {%
```

```

5254 \PackageWarning{fmtcount}%
5255 {\protect\ordinal \space already defined use
5256 \protect\FCardinal \space instead.}
5257 }

```

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

```

5258 \newrobustcmd*{\ordinalnum}[1]{%
5259 \new@ifnextchar[%
5260 {\@ordinalnum{#1}}%
5261 {\@ordinalnum{#1}[m]}%
5262 }

```

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

```

5263 \def\@ordinalnum#1[#2]{%
5264 {%
5265 \ifthenelse{\equal{#2}{f}}%
5266 {%
5267 \protect\@ordinalF{#1}{\@fc@ordstr}%
5268 }%
5269 {%
5270 \ifthenelse{\equal{#2}{n}}%
5271 {%
5272 \protect\@ordinalN{#1}{\@fc@ordstr}%
5273 }%
5274 {%
5275 \ifthenelse{\equal{#2}{m}}%
5276 {}%
5277 {%
5278 \PackageError{fmtcount}%
5279 {Invalid gender option ‘#2’}%
5280 {Available options are m, f or n}%
5281 }%
5282 \protect\@ordinalM{#1}{\@fc@ordstr}%
5283 }%
5284 }%
5285 \@fc@ordstr
5286 }%
5287 }

```

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

```

5288 \newcommand*{\storeordinal}[2]{%
5289 {%
5290 \toks0{\storeordinalnum{#1}}%
5291 \expandafter
5292 }\the\toks0\expandafter{%
5293 \the\value{#2}}%
5294 }

```

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

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

```
5295 \newrobustcmd*{\storeordinalnum}[2]{%
5296   \@ifnextchar[%
5297     {\@storeordinalnum{#1}{#2}}%
5298     {\@storeordinalnum{#1}{#2}[m]}%
5299 }
```

`storeordinalnum` Store ordinal according to gender:

```
5300 \def\@storeordinalnum#1#2[#3]{%
5301   \ifthenelse{\equal{#3}{f}}%
5302     {%
5303       \protect\@ordinalF{#2}{\@fcs@ord}
5304     }%
5305     {%
5306       \ifthenelse{\equal{#3}{n}}%
5307         {%
5308           \protect\@ordinalN{#2}{\@fcs@ord}%
5309         }%
5310         {%
5311           \ifthenelse{\equal{#3}{m}}%
5312             {}%
5313             {%
5314               \PackageError{fmtcount}%
5315                 {Invalid gender option ‘#3’}%
5316                 {Available options are m or f}%
5317             }%
5318           \protect\@ordinalM{#2}{\@fcs@ord}%
5319         }%
5320     }%
5321 \expandafter\let\csname @fcs@#1\endcsname\@fcs@ord
5322 }
```

`\FMCuse` Get stored information:

```
5323 \newcommand*{\FMCuse}[1]{\csname @fcs@#1\endcsname}
```

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

```
5324 \newcommand*{\ordinalstring}[1]{%
5325   \ordinalstringnum{\expandafter\expandafter\expandafter
5326     \the\value{#1}}%
5327 }
```

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

```
5328 \newrobustcmd*{\ordinalstringnum}[1]{%
5329   \new@ifnextchar[%
5330     {\@ordinal@string{#1}}%
5331     {\@ordinal@string{#1}[m]}%
5332 }
```

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

```
5333 \def\@ordinal@string#1[#2]{%
5334   {%
5335     \ifthenelse{\equal{#2}{f}}%
5336     {%
5337       \protect\@ordinalstringF{#1}{\@fc@ordstr}%
5338     }%
5339     {%
5340       \ifthenelse{\equal{#2}{n}}%
5341       {%
5342         \protect\@ordinalstringN{#1}{\@fc@ordstr}%
5343       }%
5344       {%
5345         \ifthenelse{\equal{#2}{m}}%
5346         {}%
5347         {%
5348           \PackageError{fmtcount}%
5349             {Invalid gender option ‘#2’ to \protect\ordinalstring}%
5350             {Available options are m, f or n}%
5351         }%
5352         \protect\@ordinalstringM{#1}{\@fc@ordstr}%
5353       }%
5354     }%
5355     \@fc@ordstr
5356   }%
5357 }
```

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

```
5358 \newcommand*\@storeordinalstring [2] {%
5359   {%
5360     \toks0{\@storeordinalstringnum{#1}}%
5361     \expandafter
5362   } \the \toks0 \expandafter {\the \value{#2}}%
5363 }
```

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

```
5364 \newrobustcmd*\@storeordinalstringnum [2] {%
5365   \@ifnextchar [%
5366   {\@store@ordinal@string{#1}{#2}}%
5367   {\@store@ordinal@string{#1}{#2}[m]}%
5368 }
```

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

```
5369 \def\@store@ordinal@string#1#2[#3]{%
5370   \ifthenelse{\equal{#3}{f}}%
5371   {%
```

```

5372   \protect\@ordinalstringF{#2}{\@fc@ordstr}%
5373 }%
5374 {%
5375   \ifthenelse{\equal{#3}{n}}%
5376   {%
5377     \protect\@ordinalstringN{#2}{\@fc@ordstr}%
5378   }%
5379   {%
5380     \ifthenelse{\equal{#3}{m}}%
5381     {}%
5382     {%
5383       \PackageError{fmtcount}%
5384       {Invalid gender option ‘#3’ to \protect\ordinalstring}%
5385       {Available options are m, f or n}%
5386     }%
5387     \protect\@ordinalstringM{#2}{\@fc@ordstr}%
5388   }%
5389 }%
5390 \expandafter\let\csname @fcs@#1\endcsname\@fc@ordstr
5391 }

```

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

```

5392 \newcommand*{\Ordinalstring}[1]{%
5393   \Ordinalstringnum{\expandafter\expandafter\expandafter\the\value{#1}}%
5394 }

```

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

```

5395 \newrobustcmd*{\Ordinalstringnum}[1]{%
5396   \new@ifnextchar[%
5397   {\@Ordinal@string{#1}}%
5398   {\@Ordinal@string{#1}[m]}%
5399 }

```

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

```

5400 \def\@Ordinal@string#1[#2]{%
5401   {%
5402     \ifthenelse{\equal{#2}{f}}%
5403     {%
5404       \protect\@OrdinalstringF{#1}{\@fc@ordstr}%
5405     }%
5406     {%
5407       \ifthenelse{\equal{#2}{n}}%
5408       {%
5409         \protect\@OrdinalstringN{#1}{\@fc@ordstr}%
5410       }%
5411       {%
5412         \ifthenelse{\equal{#2}{m}}%
5413         {}%

```



```

5414     {%
5415         \PackageError{fmtcount}%
5416         {Invalid gender option ‘#2’}%
5417         {Available options are m, f or n}%
5418     }%
5419     \protect\@OrdinalstringM{#1}{\@fc@ordstr}%
5420 }%
5421 }%
5422 \@fc@ordstr
5423 }%
5424 }

```

`\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.

```

5425 \newcommand*\storeOrdinalstring}[2]{%
5426   {%
5427     \toks0{\storeOrdinalstringnum{#1}}%
5428     \expandafter
5429   }\the\toks0\expandafter{\the\value{#2}}%
5430 }

```

`\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.

```

5431 \newrobustcmd*\storeOrdinalstringnum}[2]{%
5432   \@ifnextchar [%
5433   {\@store@Ordinal@string{#1}{#2}}%
5434   {\@store@Ordinal@string{#1}{#2}[m]}%
5435 }

```

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

```

5436 \def\@store@Ordinal@string#1#2[#3]{%
5437   \ifthenelse{\equal{#3}{f}}%
5438   {%
5439     \protect\@OrdinalstringF{#2}{\@fc@ordstr}%
5440   }%
5441   {%
5442     \ifthenelse{\equal{#3}{n}}%
5443     {%
5444       \protect\@OrdinalstringN{#2}{\@fc@ordstr}%
5445     }%
5446     {%
5447       \ifthenelse{\equal{#3}{m}}%
5448       {}%
5449     }%
5450     \PackageError{fmtcount}%
5451     {Invalid gender option ‘#3’}%
5452     {Available options are m or f}%
5453   }%
5454   \protect\@OrdinalstringM{#2}{\@fc@ordstr}%

```

```

5455 }%
5456 }%
5457 \expandafter\let\csname @fcs@#1\endcsname\@fc@ordstr
5458 }

```

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

```

5459 \newcommand*\storeORDINALstring}[2]{%
5460 {%
5461 \toks0{\storeORDINALstringnum{#1}}%
5462 \expandafter
5463 }\the\toks0\expandafter{\the\value{#2}}%
5464 }

```

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

```

5465 \newrobustcmd*\storeORDINALstringnum}[2]{%
5466 \@ifnextchar[%
5467 {\@store@ORDINAL@string{#1}{#2}}%
5468 {\@store@ORDINAL@string{#1}{#2}[m]}%
5469 }

```

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

```

5470 \def\@store@ORDINAL@string#1#2[#3]{%
5471 \ifthenelse{\equal{#3}{f}}%
5472 {%
5473 \protect\@ordinalstringF{#2}{\@fc@ordstr}%
5474 }%
5475 {%
5476 \ifthenelse{\equal{#3}{n}}%
5477 {%
5478 \protect\@ordinalstringN{#2}{\@fc@ordstr}%
5479 }%
5480 {%
5481 \ifthenelse{\equal{#3}{m}}%
5482 {}%
5483 {%
5484 \PackageError{fmtcount}%
5485 {Invalid gender option ‘#3’}%
5486 {Available options are m or f}%
5487 }%
5488 \protect\@ordinalstringM{#2}{\@fc@ordstr}%
5489 }%
5490 }%

5491 \expandafter\protected@edef\csname @fcs@#1\endcsname{%
5492 \noexpand\MakeUppercase{\@fc@ordstr}%
5493 }%
5494 }

```

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

```
5495 \newcommand*{\ORDINALstring}[1]{%
5496   \ORDINALstringnum{\expandafter\expandafter\expandafter
5497     \the\value{#1}}%
5498 }%
5499 }
```

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

```
5500 \newrobustcmd*{\ORDINALstringnum}[1]{%
5501   \new@ifnextchar[%
5502     {\@ORDINAL@string{#1}}%
5503     {\@ORDINAL@string{#1}[m]}%
5504 }
```

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

```
5505 \def\@ORDINAL@string#1[#2]{%
5506   {%
5507     \ifthenelse{\equal{#2}{f}}%
5508     {%
5509       \protect\@ordinalstringF{#1}{\@fc@ordstr}%
5510     }%
5511     {%
5512       \ifthenelse{\equal{#2}{n}}%
5513       {%
5514         \protect\@ordinalstringN{#1}{\@fc@ordstr}%
5515       }%
5516       {%
5517         \ifthenelse{\equal{#2}{m}}%
5518         {}%
5519         {%
5520           \PackageError{fmtcount}%
5521             {Invalid gender option ‘#2’}%
5522             {Available options are m, f or n}%
5523         }%
5524         \protect\@ordinalstringM{#1}{\@fc@ordstr}%
5525       }%
5526     }%
5527     \MakeUppercase{\@fc@ordstr}%
5528   }%
5529 }
```

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

```
5530 \newcommand*{\storenumberstring}[2]{%
5531   \expandafter\protect\expandafter\storenumberstringnum{#1}{%
5532     \expandafter\the\value{#2}}%
5533 }
```

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

```

5534 \newcommand{\storenumberstringnum}[2]{%
5535   \ifnextchar[%
5536     {\@store@number@string{#1}{#2}}%
5537     {\@store@number@string{#1}{#2}[m]}%
5538 }

```

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

```

5539 \def\@store@number@string#1#2[#3]{%
5540   \ifthenelse{\equal{#3}{f}}%
5541     {%
5542       \protect\@numberstringF{#2}{\@fc@numstr}%
5543     }%
5544     {%
5545       \ifthenelse{\equal{#3}{n}}%
5546         {%
5547           \protect\@numberstringN{#2}{\@fc@numstr}%
5548         }%
5549         {%
5550           \ifthenelse{\equal{#3}{m}}%
5551             {}%
5552             {%
5553               \PackageError{fmtcount}
5554                 {Invalid gender option ‘#3’}%
5555                 {Available options are m, f or n}%
5556             }%
5557           \protect\@numberstringM{#2}{\@fc@numstr}%
5558         }%
5559     }%
5560   \expandafter\let\csname @fcs@#1\endcsname\@fc@numstr
5561 }

```

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

```

5562 \newcommand*{\numberstring}[1]{%
5563   \numberstringnum{\expandafter\expandafter\expandafter
5564     \the\value{#1}}%
5565 }

```

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

```

5566 \newrobustcmd*{\numberstringnum}[1]{%
5567   \new@ifnextchar[%
5568     {\@number@string{#1}}%
5569     {\@number@string{#1}[m]}%
5570 }

```

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

```

5571 \def\@number@string#1[#2]{%
5572   {%
5573     \ifthenelse{\equal{#2}{f}}%
5574     {%

```

```

5575     \protect\@numberstringF{#1}{\@fc@numstr}%
5576 }%
5577 {%
5578     \ifthenelse{\equal{#2}{n}}%
5579     {%
5580         \protect\@numberstringN{#1}{\@fc@numstr}%
5581     }%
5582     {%
5583         \ifthenelse{\equal{#2}{m}}%
5584         {}%
5585         {%
5586             \PackageError{fmtcount}%
5587             {Invalid gender option ‘#2’}%
5588             {Available options are m, f or n}%
5589         }%
5590     }%
5591     \protect\@numberstringM{#1}{\@fc@numstr}%
5592 }%
5593 \@fc@numstr
5594 }%
5595 }

```

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

```

5596 \newcommand*\storeNumberstring[2]{%
5597   {%
5598     \toks0{\storeNumberstringnum{#1}}%
5599     \expandafter
5600   }\the\toks0\expandafter{\the\value{#2}}%
5601 }

```

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

```

5602 \newcommand{\storeNumberstringnum}[2]{%
5603   \@ifnextchar[%
5604   {\@store@Number@string{#1}{#2}}%
5605   {\@store@Number@string{#1}{#2}[m]}%
5606 }

```

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

```

5607 \def\@store@Number@string#1#2[#3]{%
5608   \ifthenelse{\equal{#3}{f}}%
5609   {%
5610     \protect\@NumberstringF{#2}{\@fc@numstr}%
5611   }%
5612   {%
5613     \ifthenelse{\equal{#3}{n}}%
5614     {%
5615       \protect\@NumberstringN{#2}{\@fc@numstr}%
5616     }%

```

```

5617   {%
5618     \ifthenelse{\equal{#3}{m}}%
5619     {}%
5620     {%
5621       \PackageError{fmtcount}%
5622       {Invalid gender option ‘#3’}%
5623       {Available options are m, f or n}%
5624     }%
5625     \protect\@NumberstringM{#2}{\@fc@numstr}%
5626   }%
5627 }%
5628 \expandafter\let\csname @fcs@#1\endcsname\@fc@numstr
5629 }

```

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

```

5630 \newcommand*{\Numberstring}[1]{%
5631   \Numberstringnum{\expandafter\expandafter\expandafter
5632     \the\value{#1}}%
5633 }

```

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

```

5634 \newrobustcmd*{\Numberstringnum}[1]{%
5635   \new@ifnextchar[%
5636     {\@Number@string{#1}}%
5637     {\@Number@string{#1}[m]}%
5638 }

```

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

```

5639 \def\@Number@string#1[#2]{%
5640   {%
5641     \ifthenelse{\equal{#2}{f}}%
5642     {%
5643       \protect\@NumberstringF{#1}{\@fc@numstr}%
5644     }%
5645     {%
5646       \ifthenelse{\equal{#2}{n}}%
5647       {%
5648         \protect\@NumberstringN{#1}{\@fc@numstr}%
5649       }%
5650       {%
5651         \ifthenelse{\equal{#2}{m}}%
5652         {}%
5653         {%
5654           \PackageError{fmtcount}%
5655           {Invalid gender option ‘#2’}%
5656           {Available options are m, f or n}%
5657         }%
5658         \protect\@NumberstringM{#1}{\@fc@numstr}%
5659       }%

```

```

5660 }%
5661 \@fc@numstr
5662 }%
5663 }

```

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

```

5664 \newcommand{\storeNUMBERstring}[2]{%
5665   {%
5666     \toks0{\storeNUMBERstringnum{#1}}%
5667     \expandafter
5668     }\the\toks0\expandafter{\the\value{#2}}%
5669 }

```

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

```

5670 \newcommand{\storeNUMBERstringnum}[2]{%
5671   \@ifnextchar[%
5672   {\@store@NUMBER@string{#1}{#2}}%
5673   {\@store@NUMBER@string{#1}{#2}[m]}%
5674 }

```

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

```

5675 \def\@store@NUMBER@string#1#2[#3]{%
5676   \ifthenelse{\equal{#3}{f}}%
5677   {%
5678     \protect\@numberstringF{#2}{\@fc@numstr}%
5679   }%
5680   {%
5681     \ifthenelse{\equal{#3}{n}}%
5682     {%
5683       \protect\@numberstringN{#2}{\@fc@numstr}%
5684     }%
5685     {%
5686       \ifthenelse{\equal{#3}{m}}%
5687       {}%
5688       {%
5689         \PackageError{fmtcount}%
5690         {Invalid gender option ‘#3’}%
5691         {Available options are m or f}%
5692       }%
5693       \protect\@numberstringM{#2}{\@fc@numstr}%
5694     }%
5695   }%
5696   \expandafter\edef\csname @fcs@#1\endcsname{%
5697     \noexpand\MakeUppercase{\@fc@numstr}%
5698   }%
5699 }

```

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

```

5700 \newcommand*{\NUMBERstring}[1]{%
5701   \NUMBERstringnum{\expandafter\expandafter\expandafter
5702     \the\value{#1}}%
5703 }

```

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

```

5704 \newrobustcmd*{\NUMBERstringnum}[1]{%
5705   \new@ifnextchar[%
5706     {\@NUMBER@string{#1}}%
5707     {\@NUMBER@string{#1}[m]}%
5708 }

```

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

```

5709 \def\@NUMBER@string#1[#2]{%
5710   {%
5711     \ifthenelse{\equal{#2}{f}}%
5712       {%
5713         \protect\@numberstringF{#1}{\@fc@numstr}%
5714       }%
5715     {%
5716       \ifthenelse{\equal{#2}{n}}%
5717         {%
5718           \protect\@numberstringN{#1}{\@fc@numstr}%
5719         }%
5720       {%
5721         \ifthenelse{\equal{#2}{m}}%
5722           {}%
5723         {%
5724           \PackageError{fmtcount}%
5725             {Invalid gender option ‘#2’}%
5726             {Available options are m, f or n}%
5727         }%
5728         \protect\@numberstringM{#1}{\@fc@numstr}%
5729       }%
5730     }%
5731     \protect\MakeUppercase{\@fc@numstr}%
5732   }%
5733 }

```

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

```

5734 \providecommand*{\binary}[1]{%
5735   \@binary{\expandafter\expandafter\expandafter
5736     \the\value{#1}}%
5737 }

```

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

```

5738 \providecommand*{\aaalph}[1]{%
5739   \@aaalph{\expandafter\expandafter\expandafter
5740     \the\value{#1}}%

```


5741 }

`\AAAAlph` As before, but upper case.

```
5742 \providecommand*\AAAAlph}[1]{%
5743   \@AAAAlph{\expandafter\expandafter\expandafter
5744     \the\value{#1}}%
5745 }
```

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

```
5746 \providecommand*\abalph}[1]{%
5747   \@abalph{\expandafter\expandafter\expandafter
5748     \the\value{#1}}%
5749 }
```

`\ABAlph` As above, but upper case.

```
5750 \providecommand*\ABAlph}[1]{%
5751   \@ABAlph{\expandafter\expandafter\expandafter
5752     \the\value{#1}}%
5753 }
```

`\hexadecimal` Hexadecimal:

```
5754 \providecommand*\hexadecimal}[1]{%
5755   \hexadecimalnum{\expandafter\expandafter\expandafter
5756     \the\value{#1}}%
5757 }
```

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

```
5758 \providecommand*\HEXADecimal}[1]{%
5759   \HEXADecimalnum{\expandafter\expandafter\expandafter
5760     \the\value{#1}}%
5761 }
5762 \newrobustcmd*\FC@Hexadecimal@warning{%
5763   \PackageWarning{fmtcount}{\string\Hexadecimal\space is deprecated, use \string\HEXADecimal\sp
5764     instead. The \string\Hexadecimal\space control sequence name is confusing as it can mislead
5765     that only the 1st letter is upper-cased.}%
5766 }
5767 \def\Hexadecimal{%
5768   \FC@Hexadecimal@warning
5769   \HEXADecimal}
```

`\octal` Octal:

```
5770 \providecommand*\octal}[1]{%
5771   \@octal{\expandafter\expandafter\expandafter
5772     \the\value{#1}}%
5773 }
```

`\decimal` Decimal:

```
5774 \providecommand*\decimal}[1]{%
```

```

5775 \@decimal{\expandafter\expandafter\expandafter
5776   \the\value{#1}}%
5777 }

```

10.4.1 Multilanguage Definitions

Flag `\fc@languagemode@detected` allows to stop scanning for multilingual mode trigger conditions. It is initialized to false as no such scanning as taken place yet.

```

5778 \newif\iffc@languagemode@detected
5779 \fc@languagemode@detectedfalse

```

`def@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.

```

5780 \def\@setdef@ultfmtcount{%
5781   \fc@languagemode@detectedtrue
5782   \ifcsundef{@ordinalMenglish}{\FCloadlang{english}}{}%
5783   \def\@ordinalstringM{\@ordinalstringMenglish}%
5784   \let\@ordinalstringF=\@ordinalstringMenglish
5785   \let\@ordinalstringN=\@ordinalstringMenglish
5786   \def\@OrdinalstringM{\@OrdinalstringMenglish}%
5787   \let\@OrdinalstringF=\@OrdinalstringMenglish
5788   \let\@OrdinalstringN=\@OrdinalstringMenglish
5789   \def\@numberstringM{\@numberstringMenglish}%
5790   \let\@numberstringF=\@numberstringMenglish
5791   \let\@numberstringN=\@numberstringMenglish
5792   \def\@NumberstringM{\@NumberstringMenglish}%
5793   \let\@NumberstringF=\@NumberstringMenglish
5794   \let\@NumberstringN=\@NumberstringMenglish
5795   \def\@ordinalM{\@ordinalMenglish}%
5796   \let\@ordinalF=\@ordinalM
5797   \let\@ordinalN=\@ordinalM
5798   \let\fmtord\fc@orddef@ult
5799 }

```

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

```

5800 \newcommand*{\fc@multiling}[2]{%
5801   \ifcsundef{@#1#2\languagename}%
5802   {% try loading it
5803     \FCloadlang{\languagename}%
5804   }%
5805   {%
5806   }%
5807   \ifcsundef{@#1#2\languagename}%
5808   {%
5809     \PackageWarning{fmtcount}%
5810     {No support for \expandafter\protect\csname #1\endcsname\space for
5811     language '\languagename'}%

```

```

5812 \ifthenelse{\equal{\language}\fc@mainlang}{%
5813 {%
5814 \FCloadlang{english}%
5815 }%
5816 {%
5817 }%
5818 \ifcsdef{@#1#2\fc@mainlang}%
5819 {%
5820 \csuse{@#1#2\fc@mainlang}%
5821 }%
5822 {%
5823 \PackageWarningNoLine{fmtcount}%
5824 {No languages loaded at all! Loading english definitions}%
5825 \FCloadlang{english}%
5826 \def\fc@mainlang{english}%
5827 \csuse{@#1#2english}%
5828 }%
5829 }%
5830 {%
5831 \csuse{@#1#2\language}%
5832 }%
5833 }

```

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

```

5834 \def\@set@multiling@fmtcount{%
5835 \fc@languagemode@detectedtrue

```

The masculine version of \numberstring:

```

5836 \def\@numberstringM{%
5837 \fc@multiling{numberstring}{M}%
5838 }%

```

The feminine version of \numberstring:

```

5839 \def\@numberstringF{%
5840 \fc@multiling{numberstring}{F}%
5841 }%

```

The neuter version of \numberstring:

```

5842 \def\@numberstringN{%
5843 \fc@multiling{numberstring}{N}%
5844 }%

```

The masculine version of \Numberstring:

```

5845 \def\@NumberstringM{%
5846 \fc@multiling{Numberstring}{M}%
5847 }%

```

The feminine version of \Numberstring:

```

5848 \def\@NumberstringF{%
5849 \fc@multiling{Numberstring}{F}%
5850 }%

```

The neuter version of \Numberstring:

```
5851 \def\@NumberstringN{%
5852   \fc@multiling{Numberstring}{N}%
5853 }%
```

The masculine version of \ordinal:

```
5854 \def\@ordinalM{%
5855   \fc@multiling{ordinal}{M}%
5856 }%
```

The feminine version of \ordinal:

```
5857 \def\@ordinalF{%
5858   \fc@multiling{ordinal}{F}%
5859 }%
```

The neuter version of \ordinal:

```
5860 \def\@ordinalN{%
5861   \fc@multiling{ordinal}{N}%
5862 }%
```

The masculine version of \ordinalstring:

```
5863 \def\@ordinalstringM{%
5864   \fc@multiling{ordinalstring}{M}%
5865 }%
```

The feminine version of \ordinalstring:

```
5866 \def\@ordinalstringF{%
5867   \fc@multiling{ordinalstring}{F}%
5868 }%
```

The neuter version of \ordinalstring:

```
5869 \def\@ordinalstringN{%
5870   \fc@multiling{ordinalstring}{N}%
5871 }%
```

The masculine version of \Ordinalstring:

```
5872 \def\@OrdinalstringM{%
5873   \fc@multiling{Ordinalstring}{M}%
5874 }%
```

The feminine version of \Ordinalstring:

```
5875 \def\@OrdinalstringF{%
5876   \fc@multiling{Ordinalstring}{F}%
5877 }%
```

The neuter version of \Ordinalstring:

```
5878 \def\@OrdinalstringN{%
5879   \fc@multiling{Ordinalstring}{N}%
5880 }%
```

Make \fmtord language dependent:

```
5881 \let\fmtord\fc@ord@multiling
5882 }
```

Check to see if babel, polyglossia, mlp, or ngerman packages have been loaded, and if yes set fmtcount in multiling. First we define some \fc@check@for@multiling macro to do such action where #1 is the package name, and #2 is a callback.

```
5883 \def\fc@check@for@multiling#1:#2\@nil{%
5884   \@ifpackageloaded{#1}{%
5885     #2\@set@multiling@fmtcount
5886   }{}%
5887 }
```

Now we define \fc@loop@on@multiling@pkg as an iterator to scan whether any of babel, polyglossia, mlp, or ngerman packages has been loaded, and if so set multilingual mode.

```
5888 \def\fc@loop@on@multiling@pkg#1,{%
5889   \def\@tempb{#1}%
5890   \ifx\@tempb\@nnil
```

We have reached the end of the loop, so stop here.

```
5891     \let\fc@loop@on@multiling@pkg\@empty
5892   \else
```

Make the \@ifpackageloaded test and break the loop if it was positive.

```
5893     \fc@check@for@multiling#1\@nil
5894     \iffc@languagemode@detected
5895     \def\fc@loop@on@multiling@pkg##1\@nil,{}%
5896     \fi
5897   \fi
5898   \fc@loop@on@multiling@pkg
5899 }
```

Now, do the loop itself, we do this at beginning of document not to constrain the order of loading fmtcount and the multilingual package babel, polyglossia, etc.:

```
5900 \AtBeginDocument{%
5901   \fc@loop@on@multiling@pkg babel:,polyglossia:,ngerman:\FCloadlang{ngerman},\@nil,
```

In the case that no multilingual package (such as babel/polyglossia/ngerman) has been loaded, then we go to multiling if a language has been loaded by package option.

```
5902   \unless\iffc@languagemode@detected\iffmtcount@language@option
```

If the multilingual mode has not been yet activated, but a language option has been passed to fmtcount, we should go to multilingual mode. However, first of, we do some sanity check, as this may help the end user understand what is wrong: we check that macro \languagename is defined, and activate the multilingual mode only then, and otherwise fall back to default legacy mode.

```
5903     \ifcsundef{languagename}%
5904     {%
5905       \PackageWarning{fmtcount}{%
5906         ‘\protect\languagename’ is undefined, you should use a language package such as bab
5907         when loading a language via package option. Reverting to default language.
5908       }%
5909       \@setdef@ultfmtcount
5910     }{%
```

5911 \@set@multiling@fmtcount

5912

Now, some more checking, having activated multilingual mode after a language option has been passed to `fmtcount`, we check that the `fmtcount` language definitions corresponding to `\language` have been loaded, and otherwise fall `\language` back to the latest `fmtcount` language definition loaded.

5913 \@FC@iflangloaded{\language}{-}{%

The current `\language` is not a `fmtcount` language that has been previously loaded. The correction is to have `\language` let to `\fc@mainlang`. Please note that, as `\iffmtcount@language@option` is true, we know that `fmtcount` has loaded some language.

5914 \PackageWarning{fmtcount}{%

5915 Setting ‘`\protect\language`’ to ‘`\fc@mainlang`’.\MessageBreak

5916 Reason is that ‘`\protect\language`’ was ‘`\language`’,\MessageBreak

5917 but ‘`\language`’ was not loaded by `fmtcount`,\MessageBreak

5918 whereas ‘`\fc@mainlang`’ was the last language loaded by `fmtcount` ;

5919 }%

5920 \let\language\fc@mainlang

5921 }%

5922 }%

5923 \else

5924 \@setdef@ultfmtcount

5925 \fi\fi

5926 }

5927 \AtBeginDocument{%

5928 \ifcsundef{FBsupR}{\let\fc@textsuperscript\textsuperscript}{\let\fc@textsuperscript\fc@fup}%

5929 }

Backwards compatibility:

5930 \let\@ordinal=\@ordinalM

5931 \let\@ordinalstring=\@ordinalstringM

5932 \let\@Ordinalstring=\@OrdinalstringM

5933 \let\@numberstring=\@numberstringM

5934 \let\@Numberstring=\@NumberstringM