

fmtcount.sty: Displaying the Values of \LaTeX Counters

Nicola L.C. Talbot

Vincent Belaïche

www.dickimaw-books.com

2020-01-30 (version 3.06)

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>	15
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-english.def</code>	20
10.1.5	<code>fc-francais.def</code>	30

10.1.6 <code>fc-french.def</code>	30
10.1.7 <code>fc-frenchb.def</code>	62
10.1.8 <code>fc-german.def</code>	62
10.1.9 <code>fc-germanb.def</code>	72
10.1.10 <code>fc-italian</code>	73
10.1.11 <code>fc-ngerman.def</code>	74
10.1.12 <code>fc-ngermanb.def</code>	74
10.1.13 <code>fc-portuges.def</code>	75
10.1.14 <code>fc-portuguese.def</code>	90
10.1.15 <code>fc-spanish.def</code>	90
10.1.16 <code>fc-UKenglish.def</code>	108
10.1.17 <code>fc-USenglish.def</code>	108
10.2 <code>fcnumparser.sty</code>	109
10.3 <code>fcprefix.sty</code>	119
10.4 <code>fmtcount.sty</code>	129
10.4.1 Multilingual Definitions	154

1 Introduction

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

2 Available Commands

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

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

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

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

is used to format the st, nd, rd, th bit. By default the ordinal is formatted as a superscript, if the package option `level` is used, it is level with the text. For example, if the current section is 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 `[\langle gender \rangle]` 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{\langle n \rangle}[\langle gender \rangle]`

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{\langle counter \rangle}[\langle gender \rangle]`

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

`\Numberstring`

`\Numberstring{\langle counter \rangle}[\langle gender \rangle]`

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

`\NUMBERstring`

`\NUMBERstring{\langle counter \rangle}[\langle gender \rangle]`

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

`\numberstringnum`

`\numberstringnum{\langle n \rangle}[\langle gender \rangle]`

`\Numberstringnum`

`\Numberstringnum{\langle n \rangle}[\langle gender \rangle]`

`\NUMBERstringnum`

`\NUMBERstringnum{\langle n \rangle}[\langle gender \rangle]`

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`).

`\ordinalstringnum`

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

`\Ordinalstringnum`

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

`\ORDINALstringnum`

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

These macros work like `\ordinalstring`, `\Ordinalstring` and `\ORDINALstring`, respectively, but take an actual number rather than a counter as the argument. For example, `\ordinalstringnum{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>]
preordinalstring	\storeordinalstring{<label>}{<counter>}[<gender>]
preOrdinalstring	\storeOrdinalstring{<label>}{<counter>}[<gender>]
preORDINALstring	\storeORDINALstring{<label>}{<counter>}[<gender>]
prenumberstring	\storenumberstring{<label>}{<counter>}[<gender>]
preNumberstring	\storeNumberstring{<label>}{<counter>}[<gender>]
preNUMBERstring	\storeNUMBERstring{<label>}{<counter>}[<gender>]
storeordinalnum	\storeordinalnum{<label>}{<number>}[<gender>]
preordinalstringnum	\storeordinalstring{<label>}{<number>}[<gender>]
preOrdinalstringnum	\storeOrdinalstringnum{<label>}{<number>}[<gender>]
preORDINALstringnum	\storeORDINALstringnum{<label>}{<number>}[<gender>]
prenumberstringnum	\storenumberstring{<label>}{<number>}[<gender>]
preNumberstringnum	\storeNumberstring{<label>}{<number>}[<gender>]
preNUMBERstringnum	\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` `\binary{<n>}`

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

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

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

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

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

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

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

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

`\HEXADecimal` `\HEXADecimal{<counter>}`

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

`\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.

`\AAAlph`

`\AAAlph{<counter>}`

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

`\aaalphnum`

`\aaalphnum{<n>}`

`\AAAlphnum`

`\AAAlphnum{<n>}`

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

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

`\abalph`

`\abalph{<counter>}`

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

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

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

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

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

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

3 Package Options

The following options can be passed to this package:

`<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 `\language`, 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`, and `italian`.

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 $\langle language \rangle$ 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 $\langle french options \rangle$ est une liste entre accolades et séparée par des virgules de réglages de la forme " $\langle clef \rangle = \langle valeur \rangle$ ", 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 $\langle dialect \rangle$ 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 $\langle clef \rangle$ s pour $\langle french options \rangle$ à utiliser sans $\langle valeur \rangle$.

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 *<boolean>* on a :

`true` pour produire des ordinaux de la forme 2^e (par défaut), ou

`false` pour produire des ordinaux de la forme 2^{ème}

vingt plural	<code>\fmtcountsetoptions{french={vingt plural=<french plural control>}}</code>
cent plural	<code>\fmtcountsetoptions{french={cent plural=<french plural control>}}</code>
mil plural	<code>\fmtcountsetoptions{french={mil plural=<french plural control>}}</code>
n-illion plural	<code>\fmtcountsetoptions{french={n-illion plural=<french plural control>}}</code>
n-illiard plural	<code>\fmtcountsetoptions{french={n-illiard plural=<french plural control>}}</code>
all plural	<code>\fmtcountsetoptions{french={all plural=<french plural control>}}</code>

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

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

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

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

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

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

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

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

$\langle x \rangle$ 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 $\langle dash or space \rangle$ 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, actuel-
 lement l'effet est le même que 1990, ou à
 always pour mettre systématiquement des traits d'union de partout.
 Par défaut, l'option vaut reformed.

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

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

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

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

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

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

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

La valeur par défaut de cette option est « le ». Il s'agit de la terminaison ajoutée à « mil » pour former le pluriel, c'est à dire « mille », cette option ne sert pas à grand chose sauf dans l'éventualité où ce pluriel serait francisé un jour — à dire vrai si cela se produisait une alternance mille/milles est plus vraisemblable, car « mille » est plus fréquent que « 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{<counter>}[<prefix options>]`

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

5 Configuration File `fmtcount.cfg`

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

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

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

and if `fmtcount.cfg` has the line:

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

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

6 LaTeX2HTML style

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

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

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

This, I agree, is an unpleasant cludge.

7 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 typeset 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\storeordinalnum{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 Dezesseis%
53 \or Dezessete%
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-english.def

English definitions

```

118\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.

```

119\newcommand*\@ordinalMenglish[2]{%
120\def\@fc@ord{}%
121\@orgargctr=#1\relax
122\@ordinalctr=#1%
123\@FCmodulo{\@ordinalctr}{100}%
124\ifnum\@ordinalctr=11\relax
125\def\@fc@ord{th}%
126\else
127\ifnum\@ordinalctr=12\relax
128\def\@fc@ord{th}%
129\else
130\ifnum\@ordinalctr=13\relax
131\def\@fc@ord{th}%
132\else
133\@FCmodulo{\@ordinalctr}{10}%
134\ifcase\@ordinalctr
135\def\@fc@ord{th}% case 0
136\or \def\@fc@ord{st}% case 1
137\or \def\@fc@ord{nd}% case 2
138\or \def\@fc@ord{rd}% case 3
139\else
140\def\@fc@ord{th}% default case
141\fi
142\fi
143\fi
144\fi
145\edef#2{\number#1\relax\noexpand\fmtord{\@fc@ord}}%
146}%
147\global\let\@ordinalMenglish\@ordinalMenglish

```

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

```

148\global\let\@ordinalFenglish=\@ordinalMenglish
149\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.

```

150\newcommand*\@@unitstringenglish[1]{%

```

```

151 \ifcase#1\relax
152     zero%
153     \or one%
154     \or two%
155     \or three%
156     \or four%
157     \or five%
158     \or six%
159     \or seven%
160     \or eight%
161     \or nine%
162 \fi
163 }%
164 \global\let\@@unitstringenglish\@@unitstringenglish

```

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

```

165 \newcommand*\@@tenstringenglish[1]{%
166     \ifcase#1\relax
167     \or ten%
168     \or twenty%
169     \or thirty%
170     \or forty%
171     \or fifty%
172     \or sixty%
173     \or seventy%
174     \or eighty%
175     \or ninety%
176 \fi
177 }%
178 \global\let\@@tenstringenglish\@@tenstringenglish

```

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

```

179 \newcommand*\@@teenstringenglish[1]{%
180     \ifcase#1\relax
181     ten%
182     \or eleven%
183     \or twelve%
184     \or thirteen%
185     \or fourteen%
186     \or fifteen%
187     \or sixteen%
188     \or seventeen%
189     \or eighteen%
190     \or nineteen%
191 \fi
192 }%
193 \global\let\@@teenstringenglish\@@teenstringenglish

```

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

```

194 \newcommand*\@@Unitstringenglish[1]{%
195     \ifcase#1\relax

```

```

196     Zero%
197     \or One%
198     \or Two%
199     \or Three%
200     \or Four%
201     \or Five%
202     \or Six%
203     \or Seven%
204     \or Eight%
205     \or Nine%
206 \fi
207}%
208\global\let\@@Unitstringenglish\@@Unitstringenglish

```

The tens:

```

209\newcommand*\@@Tenstringenglish[1]{%
210 \ifcase#1\relax
211     \or Ten%
212     \or Twenty%
213     \or Thirty%
214     \or Forty%
215     \or Fifty%
216     \or Sixty%
217     \or Seventy%
218     \or Eighty%
219     \or Ninety%
220 \fi
221}%
222\global\let\@@Tenstringenglish\@@Tenstringenglish

```

The teens:

```

223\newcommand*\@@Teenstringenglish[1]{%
224 \ifcase#1\relax
225     Ten%
226     \or Eleven%
227     \or Twelve%
228     \or Thirteen%
229     \or Fourteen%
230     \or Fifteen%
231     \or Sixteen%
232     \or Seventeen%
233     \or Eighteen%
234     \or Nineteen%
235 \fi
236}%
237\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.)

```

238 \newcommand*{\@@numberstringenglish[2]}{%
239 \ifnum#1>99999
240 \PackageError{fmtcount}{Out of range}%
241 {This macro only works for values less than 100000}%
242 \else
243 \ifnum#1<0
244 \PackageError{fmtcount}{Negative numbers not permitted}%
245 {This macro does not work for negative numbers, however
246 you can try typing "minus" first, and then pass the modulus of
247 this number}%
248 \fi
249 \fi
250 \def#2{}%
251 \@strctr=#1\relax \divide\@strctr by 1000\relax
252 \ifnum\@strctr>9
253   \divide\@strctr by 10
254   \ifnum\@strctr>1\relax
255     \let\@@fc@numstr#2\relax
256     \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
257     \@strctr=#1 \divide\@strctr by 1000\relax
258     \@FCmodulo{\@strctr}{10}%
259     \ifnum\@strctr>0\relax
260       \let\@@fc@numstr#2\relax
261       \edef#2{\@@fc@numstr-\@unitstring{\@strctr}}%
262     \fi
263   \else
264     \@strctr=#1\relax
265     \divide\@strctr by 1000\relax
266     \@FCmodulo{\@strctr}{10}%
267     \let\@@fc@numstr#2\relax
268     \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
269   \fi
270   \let\@@fc@numstr#2\relax
271   \edef#2{\@@fc@numstr\ \@thousand}%
272 \else
273   \ifnum\@strctr>0\relax
274     \let\@@fc@numstr#2\relax
275     \edef#2{\@@fc@numstr\@unitstring{\@strctr}\ \@thousand}%
276   \fi
277 \fi
278 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
279 \divide\@strctr by 100
280 \ifnum\@strctr>0\relax
281   \ifnum#1>1000\relax
282     \let\@@fc@numstr#2\relax
283     \edef#2{\@@fc@numstr\ }%
284   \fi
285   \let\@@fc@numstr#2\relax
286   \edef#2{\@@fc@numstr\@unitstring{\@strctr}\ \@hundred}%

```

```

287 \fi
288 \@strctr=#1\relax \@FCmodulo{\@strctr}{100}%
289 \ifnum#1>100\relax
290   \ifnum\@strctr>0\relax
291     \let\@fc@numstr#2\relax
292     \edef#2{\@fc@numstr \@andname\ }%
293   \fi
294 \fi
295 \ifnum\@strctr>19\relax
296   \divide\@strctr by 10\relax
297   \let\@fc@numstr#2\relax
298   \edef#2{\@fc@numstr\@tenstring{\@strctr}}%
299   \@strctr=#1\relax \@FCmodulo{\@strctr}{10}%
300   \ifnum\@strctr>0\relax
301     \let\@fc@numstr#2\relax
302     \edef#2{\@fc@numstr-\@unitstring{\@strctr}}%
303   \fi
304 \else
305   \ifnum\@strctr<10\relax
306     \ifnum\@strctr=0\relax
307       \ifnum#1<100\relax
308         \let\@fc@numstr#2\relax
309         \edef#2{\@fc@numstr\@unitstring{\@strctr}}%
310       \fi
311     \else
312       \let\@fc@numstr#2\relax
313       \edef#2{\@fc@numstr\@unitstring{\@strctr}}%
314     \fi
315   \else
316     \@FCmodulo{\@strctr}{10}%
317     \let\@fc@numstr#2\relax
318     \edef#2{\@fc@numstr\@teenstring{\@strctr}}%
319   \fi
320 \fi
321 }%
322 \global\let\@numberstringenglish\@numberstringenglish

```

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

```

323 \newcommand*{\@numberstringMenglish}[2]{%
324   \let\@unitstring=\@unitstringenglish
325   \let\@teenstring=\@teenstringenglish
326   \let\@tenstring=\@tenstringenglish
327   \def\@hundred{hundred}\def\@thousand{thousand}%
328   \def\@andname{and}%
329   \@numberstringenglish{#1}{#2}%
330 }%
331 \global\let\@numberstringMenglish\@numberstringMenglish

```

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

```

332 \global\let\@numberstringFenglish=\@numberstringMenglish

```



```
333 \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.

```
334 \newcommand* \@NumberstringMenglish[2] {%
335   \let \@unitstring=\@Unitstringenglish
336   \let \@teenstring=\@Teenstringenglish
337   \let \@tenstring=\@Tenstringenglish
338   \def \@hundred{Hundred}\def \@thousand{Thousand}%
339   \def \@andname{and}%
340   \@numberstringenglish{#1}{#2}%
341 }%
342 \global\let \@NumberstringMenglish \@NumberstringMenglish
```

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

```
343 \global\let \@NumberstringFenglish=\@NumberstringMenglish
344 \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:

```
345 \newcommand* \@unitthstringenglish[1] {%
346   \ifcase#1\relax
347     zeroth%
348   \or first%
349   \or second%
350   \or third%
351   \or fourth%
352   \or fifth%
353   \or sixth%
354   \or seventh%
355   \or eighth%
356   \or ninth%
357   \fi
358 }%
359 \global\let \@unitthstringenglish \@unitthstringenglish
```

Next the tens:

```
360 \newcommand* \@tenthstringenglish[1] {%
361   \ifcase#1\relax
362     \or tenth%
363     \or twentieth%
364     \or thirtieth%
365     \or fortieth%
366     \or fiftieth%
367     \or sixtieth%
368     \or seventieth%
369     \or eightieth%
370     \or ninetieth%
371   \fi
372 }%
373 \global\let \@tenthstringenglish \@tenthstringenglish
```

The teens:

```
374 \newcommand*{\@@teenthstringenglish[1]}{%
375   \ifcase#1\relax
376     tenth%
377     \or eleventh%
378     \or twelfth%
379     \or thirteenth%
380     \or fourteenth%
381     \or fifteenth%
382     \or sixteenth%
383     \or seventeenth%
384     \or eighteenth%
385     \or nineteenth%
386   \fi
387 }%
388 \global\let\@@teenthstringenglish\@@teenthstringenglish
```

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

```
389 \newcommand*{\@@Unitthstringenglish[1]}{%
390   \ifcase#1\relax
391     Zeroth%
392     \or First%
393     \or Second%
394     \or Third%
395     \or Fourth%
396     \or Fifth%
397     \or Sixth%
398     \or Seventh%
399     \or Eighth%
400     \or Ninth%
401   \fi
402 }%
403 \global\let\@@Unitthstringenglish\@@Unitthstringenglish
```

The tens:

```
404 \newcommand*{\@@Tenthstringenglish[1]}{%
405   \ifcase#1\relax
406     Tenth%
407     \or Twentieth%
408     \or Thirtieth%
409     \or Fortieth%
410     \or Fiftieth%
411     \or Sixtieth%
412     \or Seventieth%
413     \or Eightieth%
414     \or Ninetieth%
415   \fi
416 }%
417 \global\let\@@Tenthstringenglish\@@Tenthstringenglish
```

The teens:

```
418 \newcommand*{\@@Teenthstringenglish[1]}{%
```

```
419   \ifcase#1\relax
```

```
420     Tenth%
```

```
421     \or Eleventh%
```

```
422     \or Twelfth%
```

```
423     \or Thirteenth%
```

```
424     \or Fourteenth%
```

```
425     \or Fifteenth%
```

```
426     \or Sixteenth%
```

```
427     \or Seventeenth%
```

```
428     \or Eighteenth%
```

```
429     \or Nineteenth%
```

```
430   \fi
```

```
431 }%
```

```
432 \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.

```
433 \newcommand*{\@@ordinalstringenglish[2]}{%
```

```
434   \@strctr=#1\relax
```

```
435   \ifnum#1>99999
```

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

```
437   {This macro only works for values less than 100000 (value given: \number\@strctr)}%
```

```
438   \else
```

```
439   \ifnum#1<0
```

```
440   \PackageError{fmtcount}{Negative numbers not permitted}%
```

```
441   {This macro does not work for negative numbers, however
```

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

```
443   this number}%
```

```
444   \fi
```

```
445   \def#2{}%
```

```
446   \fi
```

```
447   \@strctr=#1\relax \divide\@strctr by 1000\relax
```

```
448   \ifnum\@strctr>9\relax
```

#1 is greater or equal to 10000

```
449   \divide\@strctr by 10
```

```
450   \ifnum\@strctr>1\relax
```

```
451     \let\@@fc@ordstr#2\relax
```

```
452     \edef#2{\@@fc@ordstr\@tenstring{\@strctr}}%
```

```
453     \@strctr=#1\relax
```

```
454     \divide\@strctr by 1000\relax
```

```
455     \@FCmodulo{\@strctr}{10}%
```

```
456     \ifnum\@strctr>0\relax
```

```
457       \let\@@fc@ordstr#2\relax
```

```
458       \edef#2{\@@fc@ordstr-\@unitstring{\@strctr}}%
```

```
459     \fi
```

```
460   \else
```

```
461     \@strctr=#1\relax \divide\@strctr by 1000\relax
```

```

462 \FCmodulo{\@strctr}{10}%
463 \let\@@fc@ordstr#2\relax
464 \edef#2{\@@fc@ordstr\@teenstring{\@strctr}}%
465 \fi
466 \@strctr=#1\relax \FCmodulo{\@strctr}{1000}%
467 \ifnum\@strctr=0\relax
468 \let\@@fc@ordstr#2\relax
469 \edef#2{\@@fc@ordstr\ \@thousandth}%
470 \else
471 \let\@@fc@ordstr#2\relax
472 \edef#2{\@@fc@ordstr\ \@thousand}%
473 \fi
474 \else
475 \ifnum\@strctr>0\relax
476 \let\@@fc@ordstr#2\relax
477 \edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
478 \@strctr=#1\relax \FCmodulo{\@strctr}{1000}%
479 \let\@@fc@ordstr#2\relax
480 \ifnum\@strctr=0\relax
481 \edef#2{\@@fc@ordstr\ \@thousandth}%
482 \else
483 \edef#2{\@@fc@ordstr\ \@thousand}%
484 \fi
485 \fi
486 \fi
487 \@strctr=#1\relax \FCmodulo{\@strctr}{1000}%
488 \divide\@strctr by 100
489 \ifnum\@strctr>0\relax
490 \ifnum#1>1000\relax
491 \let\@@fc@ordstr#2\relax
492 \edef#2{\@@fc@ordstr\ }%
493 \fi
494 \let\@@fc@ordstr#2\relax
495 \edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
496 \@strctr=#1\relax \FCmodulo{\@strctr}{100}%
497 \let\@@fc@ordstr#2\relax
498 \ifnum\@strctr=0\relax
499 \edef#2{\@@fc@ordstr\ \@hundredth}%
500 \else
501 \edef#2{\@@fc@ordstr\ \@hundred}%
502 \fi
503 \fi
504 \@strctr=#1\relax \FCmodulo{\@strctr}{100}%
505 \ifnum#1>100\relax
506 \ifnum\@strctr>0\relax
507 \let\@@fc@ordstr#2\relax
508 \edef#2{\@@fc@ordstr\ \@andname\ }%
509 \fi
510 \fi

```

```

511 \ifnum \@strctr>19\relax
512   \@tmpstrctr=\@strctr
513   \divide \@strctr by 10\relax
514   \@FCmodulo{\@tmpstrctr}{10}%
515   \let \@fc@ordstr#2\relax
516   \ifnum \@tmpstrctr=0\relax
517     \edef#2{\@fc@ordstr\@tenthstring{\@strctr}}%
518   \else
519     \edef#2{\@fc@ordstr\@tenstring{\@strctr}}%
520   \fi
521   \@strctr=#1\relax \@FCmodulo{\@strctr}{10}%
522   \ifnum \@strctr>0\relax
523     \let \@fc@ordstr#2\relax
524     \edef#2{\@fc@ordstr-\@unitthstring{\@strctr}}%
525   \fi
526 \else
527   \ifnum \@strctr<10\relax
528     \ifnum \@strctr=0\relax
529       \ifnum#1<100\relax
530         \let \@fc@ordstr#2\relax
531         \edef#2{\@fc@ordstr\@unitthstring{\@strctr}}%
532       \fi
533     \else
534       \let \@fc@ordstr#2\relax
535       \edef#2{\@fc@ordstr\@unitthstring{\@strctr}}%
536     \fi
537   \else
538     \@FCmodulo{\@strctr}{10}%
539     \let \@fc@ordstr#2\relax
540     \edef#2{\@fc@ordstr\@teenthstring{\@strctr}}%
541   \fi
542 \fi
543 }%
544 \global\let \@ordinalstringenglish \@ordinalstringenglish

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

545 \newcommand*{\@ordinalstringMenglish}[2]{%
546   \let \@unitthstring=\@unitthstringenglish
547   \let \@teenthstring=\@teenthstringenglish
548   \let \@tenthstring=\@tenthstringenglish
549   \let \@unitstring=\@unitstringenglish
550   \let \@teenstring=\@teenstringenglish
551   \let \@tenstring=\@tenstringenglish
552   \def \@andname{and}%
553   \def \@hundred{hundred}\def \@thousand{thousand}%
554   \def \@hundredth{hundredth}\def \@thousandth{thousandth}%
555   \@ordinalstringenglish{#1}{#2}%
556 }%
557 \global\let \@ordinalstringMenglish \@ordinalstringMenglish

```

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

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

First letter of each word in upper case:

```
560 \newcommand*{\@OrdinalstringMenglish}[2]{%
561   \let\@unitthstring=\@Unitthstringenglish
562   \let\@teenthstring=\@Teenthstringenglish
563   \let\@tenthstring=\@Tenthstringenglish
564   \let\@unitstring=\@Unitstringenglish
565   \let\@teenstring=\@Teenstringenglish
566   \let\@tenstring=\@Tenstringenglish
567   \def\@andname{and}%
568   \def\@hundred{Hundred}\def\@thousand{Thousand}%
569   \def\@hundredth{Hundredth}\def\@thousandth{Thousandth}%
570   \@ordinalstringenglish{#1}{#2}%
571 }%
572 \global\let\@OrdinalstringMenglish\@OrdinalstringMenglish
```

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

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

10.1.5 fc-francais.def

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

Set francais to be equivalent to french.

```
577 \global\let\@ordinalMfrancais=\@ordinalMfrench
578 \global\let\@ordinalFfrancais=\@ordinalFfrench
579 \global\let\@ordinalNfrancais=\@ordinalNfrench
580 \global\let\@numberstringMfrancais=\@numberstringMfrench
581 \global\let\@numberstringFfrancais=\@numberstringFfrench
582 \global\let\@numberstringNfrancais=\@numberstringNfrench
583 \global\let\@NumberstringMfrancais=\@NumberstringMfrench
584 \global\let\@NumberstringFfrancais=\@NumberstringFfrench
585 \global\let\@NumberstringNfrancais=\@NumberstringNfrench
586 \global\let\@ordinalstringMfrancais=\@ordinalstringMfrench
587 \global\let\@ordinalstringFfrancais=\@ordinalstringFfrench
588 \global\let\@ordinalstringNfrancais=\@ordinalstringNfrench
589 \global\let\@OrdinalstringMfrancais=\@OrdinalstringMfrench
590 \global\let\@OrdinalstringFfrancais=\@OrdinalstringFfrench
591 \global\let\@OrdinalstringNfrancais=\@OrdinalstringNfrench
```

10.1.6 fc-french.def

Definitions for French.

```
592 \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 developped 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.

```
593 \ifcsundef{fc@gl@let}{\global\let\fc@gl@let\let}{\PackageError{fmtcount}{Command already defined}}
594 \protect\fc@gl@let\space already defined.}
595 \ifcsundef{fc@gl@def}{\global\let\fc@gl@def\def}{\PackageError{fmtcount}{Command already defined}}
596 \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'.
597 \gdef\fc@french@set@plural#1#2#3#4#5#6{%
598   \ifthenelse{\equal{#2}{reformed}}{%
599     \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{#3}%
600   }{%
601     \ifthenelse{\equal{#2}{traditional}}{%
602       \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{#4}%
603     }{%
604       \ifthenelse{\equal{#2}{reformed o}}{%
605         \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{#5}%
606       }{%
607         \ifthenelse{\equal{#2}{traditional o}}{%
608           \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{#6}%
609         }{%
610           \ifthenelse{\equal{#2}{always}}{%
611             \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{0}%
612           }{%
613             \ifthenelse{\equal{#2}{never}}{%
614               \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{1}%
615             }{%
616               \ifthenelse{\equal{#2}{multiple}}{%
617                 \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{2}%
618               }{%
619                 \ifthenelse{\equal{#2}{multiple g-last}}{%
620                   \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{3}%
621                 }{%
622                   \ifthenelse{\equal{#2}{multiple l-last}}{%
623                     \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{4}%
624                   }{%
```

```

625 \ifthenelse{\equal{#2}{multiple lng-last}}{%
626 \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{5}%
627 }{%
628 \ifthenelse{\equal{#2}{multiple ng-last}}{%
629 \expandafter\fc@gl@def\csname fc@frenchoptions@#1@plural\endcsname{6}%
630 }{%
631 \PackageError{fmtcount}{Unexpected argument}{%
632 '#2' was unexpected: french option '#1 plural' expects 'reformed', 't
633 'reformed o', 'traditional o', 'always', 'never', 'multiple', 'multip
634 'multiple l-last', 'multiple lng-last', or 'multiple ng-last'.%
635 }}}}}}}}}}}}}%

```

Now a shorthand \@tempa is defined just to define all the options controlling plural mark. This shorthand takes into account that 'reformed' and 'traditional' have the same effect, and so do 'reformed o' and 'traditional o'.

```

636 \def\@tempa#1#2#3{%
637 \define@key{fcfrench}{#1 plural}[reformed]{%
638 \fc@french@set@plural{#1}{##1}{#2}{#2}{#3}{#3}%
639 }%

```

Macro \@tempb takes a macro as argument, and makes its current definition global. Like here it is useful when the macro name contains non-letters, and we have to resort to the \csname...\endcsname construct.

```

640 \expandafter\@tempb\csname KV@fcfrench@#1 plural\endcsname
641 }%
642 \def\@tempb#1{%
643 \global\let#1#1
644 }%
645 \@tempa{vingt}{4}{5}
646 \@tempa{cent}{4}{5}
647 \@tempa{mil}{0}{0}
648 \@tempa{n-illion}{2}{6}
649 \@tempa{n-illiard}{2}{6}

```

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

```

650 \define@key{fcfrench}{all plural}[reformed]{%
651 \csname KV@fcfrench@vingt plural\endcsname{#1}%
652 \csname KV@fcfrench@cent plural\endcsname{#1}%
653 \csname KV@fcfrench@mil plural\endcsname{#1}%
654 \csname KV@fcfrench@n-illion plural\endcsname{#1}%
655 \csname KV@fcfrench@n-illiard plural\endcsname{#1}%
656 }%
657 \expandafter\@tempb\csname KV@fcfrench@all plural\endcsname

```

Now options 'dash or space', we have three possible key values:

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

```

658 \define@key{fcfrench}{dash or space}[reformed]{%
659   \ifthenelse{\equal{#1}{traditional}}{%
660     \let\fc@frenchoptions@supermillion@dos\space%
661     \let\fc@frenchoptions@submillion@dos\space
662   }{%
663     \ifthenelse{\equal{#1}{reformed}\or\equal{#1}{1990}}{%
664       \let\fc@frenchoptions@supermillion@dos\space
665       \def\fc@frenchoptions@submillion@dos{-}%
666     }{%
667       \ifthenelse{\equal{#1}{always}}{%
668         \def\fc@frenchoptions@supermillion@dos{-}%
669         \def\fc@frenchoptions@submillion@dos{-}%
670       }{%
671         \PackageError{fmtcount}{Unexpected argument}{%
672           French option ‘dash or space’ expects ‘always’, ‘reformed’ or ‘traditional’
673         }
674       }%
675     }%
676   }%
677 }%
```

Option ‘scale’, can take 3 possible values:

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

```

678 \define@key{fcfrench}{scale}[recursive]{%
679   \ifthenelse{\equal{#1}{long}}{%
680     \let\fc@poweroften\fc@@pot@longscalefrench
681   }{%
682     \ifthenelse{\equal{#1}{recursive}}{%
683       \let\fc@poweroften\fc@@pot@recursivefrench
684     }{%
685       \ifthenelse{\equal{#1}{short}}{%
686         \let\fc@poweroften\fc@@pot@shortscalefrench
687       }{%
688         \PackageError{fmtcount}{Unexpected argument}{%
689           French option ‘scale’ expects ‘long’, ‘recursive’ or ‘short’
690         }
691       }%
692     }%
693   }%
694 }%
```

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”

```

695 \define@key{fcfrench}{n-illiard upto}[infinity]{%
696   \ifthenelse{\equal{#1}{infinity}}{%
697     \def\fc@longscale@nilliard@upto{0}%
698   }{%
699     \ifthenelse{\equal{#1}{infty}}{%
700       \def\fc@longscale@nilliard@upto{0}%
701     }{%
702       \if Q\ifnum9<1#1Q\fi\else
703         \PackageError{fmtcount}{Unexpected argument}{%
704           French option ‘milliard threshold’ expects ‘infinity’, or equivalently ‘infty’, or a no
705           integer.}%
706       \fi
707       \def\fc@longscale@nilliard@upto{#1}%
708     }%
709 }%
```

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.

```

710 \def\@tempa#1{%
711   \define@key{fcfrench}{#1}[]{%
712     \PackageError{fmtcount}{Unexpected argument}{French option with key ‘#1’ does not take
713     any value}}%
714   \csgdef{KV@fcfrench@#1@default}{%
715     \fc@gl@def\fmtcount@french{#1}}%
716 }%
```

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

```

718 \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.

```

719 \define@key{fcfrench}{dialect}[france]{%
720   \ifthenelse{\equal{#1}{france}
721     \or\equal{#1}{swiss}
722     \or\equal{#1}{belgian}}{%
723     \def\fmtcount@french{#1}}{%
724     \PackageError{fmtcount}{Invalid value ‘#1’ to french option dialect key}
725     {Option ‘french’ can only take the values ‘france’,
726     ‘belgian’ or ‘swiss’}}%
727 \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’.

```

728 \define@key{fcfrench}{mil plural mark}[1e]{%
729   \def\fc@frenchoptions@mil@plural@mark{#1}}
730 \expandafter\@tempb\csname KV@fcfrench@mil plural mark\endcsname

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

\fc@UpperCaseFirstLetter The macro \fc@UpperCaseFirstLetter is such that \fc@UpperCaseFirstLetter<word>\@nil
expands to \word with first letter capitalized and remainder unchanged.
731 \gdef\fc@UpperCaseFirstLetter#1#2\@nil{%
732   \uppercase{#1}#2}

\fc@CaseIden The macro \fc@CaseIden is such that \fc@CaseIden<word>\@nil expands to \word un-
changed.
733 \gdef\fc@CaseIden#1\@nil{%
734   #1%
735 }%

\fc@UpperCaseAll The macro \fc@UpperCaseAll is such that \fc@UpperCaseAll<word>\@nil expands to
\word all capitalized.
736 \gdef\fc@UpperCaseAll#1\@nil{%
737   \uppercase{#1}%
738 }%

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

\fc@gcase The macro \fc@gcase is the capitalizing macro for global (the completed number) capital-
ization. By default we set it to identity, ie. no capitalization.
740 \global\let\fc@gcase\fc@CaseIden

\fc@apply@gcase The macro \fc@apply@gcase simply applies \fc@gcase to \@tempa, knowing that \@tempa
is the macro containing the result of formatting.
741 \gdef\fc@apply@gcase{%
  First of all we expand whatever \fc@wcase...\@nil found within \@tempa.
742   \protected@edef\@tempa{\@tempa}%
743   \protected@edef\@tempa{\expandafter\fc@gcase\@tempa\@nil}%
744 }

\ordinalMfrench
745 \newcommand*{\@ordinalMfrench}[2]{%
746   \iffmtord@abbrv
747   \ifnum#1=1 %
748     \edef#2{\number#1\relax\noexpand\fmtord{er}}%
749   \else
750     \edef#2{\number#1\relax\noexpand\fmtord{e}}%
751   \fi
752 \else
753   \PackageWarning{fmtcount}{Non abbreviated ordinal finals ('eme) are
754     considered incorrect in French.}%

```

```

755 \ifnum#1=1 %
756   \edef#2{\number#1\relax\noexpand\fmtord{er}}%
757 \else
758   \protected@edef#2{\number#1\relax\noexpand\fmtord{\protect\'eme}}%
759 \fi
760 \fi}
761 \global\let\@ordinalMfrench\@ordinalMfrench

```

@ordinalFfrench

```

762 \newcommand*{\@ordinalFfrench}[2]{%
763 \iffmtord@abbrev
764   \ifnum#1=1 %
765     \edef#2{\number#1\relax\noexpand\fmtord{re}}%
766   \else
767     \edef#2{\number#1\relax\noexpand\fmtord{e}}%
768   \fi
769 \else
770   \PackageWarning{fmtcount}{Non abbreviated ordinal finals ('eme) are
771     considered incorrect in French.}%
772   \ifnum#1=1 %
773     \protected@edef#2{\number#1\relax\noexpand\fmtord{\protect\'ere}}%
774   \else
775     \protected@edef#2{\number#1\relax\noexpand\fmtord{\protect\'eme}}%
776   \fi
777 \fi}
778 \global\let\@ordinalFfrench\@ordinalFfrench

```

In French neutral gender and masculine gender are formally identical.

```

779 \global\let\@ordinalNfrench\@ordinalMfrench

```

unitstringfrench

```

780 \newcommand*{\@unitstringfrench}[1]{%
781 \noexpand\fc@wcase
782 \ifcase#1 %
783 z\'ero%
784 \or un%
785 \or deux%
786 \or trois%
787 \or quatre%
788 \or cinq%
789 \or six%
790 \or sept%
791 \or huit%
792 \or neuf%
793 \fi
794 \noexpand\@nil
795 }%
796 \global\let\@unitstringfrench\@unitstringfrench

```

tenstringfrench

```

797 \newcommand*{\@tenstringfrench}[1]{%

```

```

798 \noexpand\fc@wcase
799 \ifcase#1 %
800 \or dix%
801 \or vingt%
802 \or trente%
803 \or quarante%
804 \or cinquante%
805 \or soixante%
806 \or septante%
807 \or huitante%
808 \or nonante%
809 \or cent%
810 \fi
811 \noexpand\@nil
812 }%
813 \global\let\@@tenstringfrench\@@tenstringfrench

```

teenstringfrench

```

814 \newcommand*\@@teenstringfrench}[1]{%
815 \noexpand\fc@wcase
816 \ifcase#1 %
817     dix%
818 \or onze%
819 \or douze%
820 \or treize%
821 \or quatorze%
822 \or quinze%
823 \or seize%
824 \or dix\noexpand\@nil-\noexpand\fc@wcase sept%
825 \or dix\noexpand\@nil-\noexpand\fc@wcase huit%
826 \or dix\noexpand\@nil-\noexpand\fc@wcase neuf%
827 \fi
828 \noexpand\@nil
829 }%
830 \global\let\@@teenstringfrench\@@teenstringfrench

```

seventiesfrench

```

831 \newcommand*\@@seventiesfrench}[1]{%
832 \@tenstring{6}%
833 \ifnum#1=1 %
834 \fc@frenchoptions@submillion@dos\@andname\fc@frenchoptions@submillion@dos
835 \else
836 -%
837 \fi
838 \@teenstring{#1}%
839 }%
840 \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".

```

841 \newcommand*{\@@eightiesfrench[1]}{
842 \fc@wcase quatre\@nil-\noexpand\fc@wcase vingt%
843 \ifnum#1>0 %
844   \ifnum\fc@frenchoptions@vingt@plural=0 % vingt plural=always
845   s%
846   \fi
847   \noexpand\@nil
848   -\@unitstring{#1}%
849 \else
850   \ifcase\fc@frenchoptions@vingt@plural\space
851     s% 0: always
852   \or
853     % 1: never
854   \or
855     s% 2: multiple
856   \or
857     % 3: multiple g-last
858     \ifnum\count0=\count6\ifnum\count9=0 s\fi\fi
859   \or
860     % 4: multiple l-last
861     \ifnum\count9=1 %
862     \else
863       s%
864     \fi
865   \or
866     % 5: multiple lng-last
867     \ifnum\count9=1 %
868     \else
869       \ifnum\count0>0 %
870       s%
871     \fi
872   \fi
873   \or
874     % or 6: multiple ng-last
875     \ifnum\count0>0 %
876     s%
877   \fi
878   \fi
879   \noexpand\@nil
880 \fi
881 }%
```

```

882\global\let\@@eightiesfrench\@@eightiesfrench
883\newcommand*\@@ninetiesfrench}[1]{%
884\fc@wcase quatre\@nil-\noexpand\fc@wcase vingt%
885\ifnum\fc@frenchoptions@vingt@plural=0 % vingt plural=always
886 s%
887\fi
888\noexpand\@nil
889-\@teenstring{#1}%
890}%
891\global\let\@@ninetiesfrench\@@ninetiesfrench
892\newcommand*\@@seventiesfrenchswiss}[1]{%
893\@tenstring{7}%
894\ifnum#1=1\ \@andname\ \fi
895\ifnum#1>1-\fi
896\ifnum#1>0 \@unitstring{#1}\fi
897}%
898\global\let\@@seventiesfrenchswiss\@@seventiesfrenchswiss
899\newcommand*\@@eightiesfrenchswiss}[1]{%
900\@tenstring{8}%
901\ifnum#1=1\ \@andname\ \fi
902\ifnum#1>1-\fi
903\ifnum#1>0 \@unitstring{#1}\fi
904}%
905\global\let\@@eightiesfrenchswiss\@@eightiesfrenchswiss
906\newcommand*\@@ninetiesfrenchswiss}[1]{%
907\@tenstring{9}%
908\ifnum#1=1\ \@andname\ \fi
909\ifnum#1>1-\fi
910\ifnum#1>0 \@unitstring{#1}\fi
911}%
912\global\let\@@ninetiesfrenchswiss\@@ninetiesfrenchswiss

```

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

```

913\newcommand*\fc@french@common{%
914 \let\fc@wcase\fc@CaseIden
915 \let\@unitstring=\@unitstringfrench
916 \let\@teenstring=\@teenstringfrench
917 \let\@tenstring=\@tenstringfrench
918 \def\@hundred{cent}%
919 \def\@andname{et}%
920}%
921\global\let\fc@french@common\fc@french@common
922\newcommand*\@numberstringMfrenchswiss}[2]{%
923\fc@french@common
924\let\fc@gcase\fc@CaseIden
925\let\@seventies=\@seventiesfrenchswiss
926\let\@eighties=\@eightiesfrenchswiss
927\let\@nineties=\@ninetiesfrenchswiss

```

```

928 \let\fc@nbrstr@preamble\@empty
929 \let\fc@nbrstr@postamble\@empty
930 \@@numberstringfrench{#1}{#2}}
931 \global\let\@numberstringMfrenchswiss\@numberstringMfrenchswiss
932 \newcommand*{\@numberstringMfrenchfrance}[2]{%
933 \fc@french@common
934 \let\fc@gcase\fc@CaseIden
935 \let\@seventies=\@seventiesfrench
936 \let\@eighties=\@eightiesfrench
937 \let\@nineties=\@ninetiesfrench
938 \let\fc@nbrstr@preamble\@empty
939 \let\fc@nbrstr@postamble\@empty
940 \@@numberstringfrench{#1}{#2}}
941 \global\let\@numberstringMfrenchfrance\@numberstringMfrenchfrance
942 \newcommand*{\@numberstringMfrenchbelgian}[2]{%
943 \fc@french@common
944 \let\fc@gcase\fc@CaseIden
945 \let\@seventies=\@seventiesfrenchswiss
946 \let\@eighties=\@eightiesfrench
947 \let\@nineties=\@ninetiesfrench
948 \let\fc@nbrstr@preamble\@empty
949 \let\fc@nbrstr@postamble\@empty
950 \@@numberstringfrench{#1}{#2}}
951 \global\let\@numberstringMfrenchbelgian\@numberstringMfrenchbelgian
952 \let\@numberstringMfrench=\@numberstringMfrenchfrance
953 \newcommand*{\@numberstringFfrenchswiss}[2]{%
954 \fc@french@common
955 \let\fc@gcase\fc@CaseIden
956 \let\@seventies=\@seventiesfrenchswiss
957 \let\@eighties=\@eightiesfrenchswiss
958 \let\@nineties=\@ninetiesfrenchswiss
959 \let\fc@nbrstr@preamble\fc@nbrstr@Fpreamble
960 \let\fc@nbrstr@postamble\@empty
961 \@@numberstringfrench{#1}{#2}}
962 \global\let\@numberstringFfrenchswiss\@numberstringFfrenchswiss
963 \newcommand*{\@numberstringFfrenchfrance}[2]{%
964 \fc@french@common
965 \let\fc@gcase\fc@CaseIden
966 \let\@seventies=\@seventiesfrench
967 \let\@eighties=\@eightiesfrench
968 \let\@nineties=\@ninetiesfrench
969 \let\fc@nbrstr@preamble\fc@nbrstr@Fpreamble
970 \let\fc@nbrstr@postamble\@empty
971 \@@numberstringfrench{#1}{#2}}
972 \global\let\@numberstringFfrenchfrance\@numberstringFfrenchfrance
973 \newcommand*{\@numberstringFfrenchbelgian}[2]{%
974 \fc@french@common
975 \let\fc@gcase\fc@CaseIden
976 \let\@seventies=\@seventiesfrenchswiss

```



```

977 \let\@eighties=\@@eightiesfrench
978 \let\@nineties=\@@ninetiesfrench
979 \let\fc@nbrstr@preamble\fc@nbrstr@Fpreamble
980 \let\fc@nbrstr@postamble\@empty
981 \@@numberstringfrench{#1}{#2}}
982 \global\let\@numberstringFfrenchbelgian\@numberstringFfrenchbelgian
983 \global\let\@numberstringFfrench=\@numberstringFfrenchfrance
984 \global\let\@ordinalstringNfrench\@ordinalstringMfrench
985 \newcommand*{\@NumberstringMfrenchswiss}[2]{%
986 \fc@french@common
987 \let\fc@gcase\fc@UpperCaseFirstLetter
988 \let\@seventies=\@@seventiesfrenchswiss
989 \let\@eighties=\@@eightiesfrenchswiss
990 \let\@nineties=\@@ninetiesfrenchswiss
991 \let\fc@nbrstr@preamble\@empty
992 \let\fc@nbrstr@postamble\fc@apply@gcase
993 \@@numberstringfrench{#1}{#2}}
994 \global\let\@NumberstringMfrenchswiss\@NumberstringMfrenchswiss
995 \newcommand*{\@NumberstringMfrenchfrance}[2]{%
996 \fc@french@common
997 \let\fc@gcase\fc@UpperCaseFirstLetter
998 \let\@seventies=\@@seventiesfrench
999 \let\@eighties=\@@eightiesfrench
1000 \let\@nineties=\@@ninetiesfrench
1001 \let\fc@nbrstr@preamble\@empty
1002 \let\fc@nbrstr@postamble\fc@apply@gcase
1003 \@@numberstringfrench{#1}{#2}}
1004 \global\let\@NumberstringMfrenchfrance\@NumberstringMfrenchfrance
1005 \newcommand*{\@NumberstringMfrenchbelgian}[2]{%
1006 \fc@french@common
1007 \let\fc@gcase\fc@UpperCaseFirstLetter
1008 \let\@seventies=\@@seventiesfrenchswiss
1009 \let\@eighties=\@@eightiesfrench
1010 \let\@nineties=\@@ninetiesfrench
1011 \let\fc@nbrstr@preamble\@empty
1012 \let\fc@nbrstr@postamble\fc@apply@gcase
1013 \@@numberstringfrench{#1}{#2}}
1014 \global\let\@NumberstringMfrenchbelgian\@NumberstringMfrenchbelgian
1015 \global\let\@NumberstringMfrench=\@NumberstringMfrenchfrance
1016 \newcommand*{\@NumberstringFfrenchswiss}[2]{%
1017 \fc@french@common
1018 \let\fc@gcase\fc@UpperCaseFirstLetter
1019 \let\@seventies=\@@seventiesfrenchswiss
1020 \let\@eighties=\@@eightiesfrenchswiss
1021 \let\@nineties=\@@ninetiesfrenchswiss
1022 \let\fc@nbrstr@preamble\fc@nbrstr@Fpreamble
1023 \let\fc@nbrstr@postamble\fc@apply@gcase
1024 \@@numberstringfrench{#1}{#2}}
1025 \global\let\@NumberstringFfrenchswiss\@NumberstringFfrenchswiss

```

```

1026 \newcommand*{\@NumberstringFfrenchfrance}[2]{%
1027 \fc@french@common
1028 \let\fc@gcase\fc@UpperCaseFirstLetter
1029 \let\@seventies=\@seventiesfrench
1030 \let\@eighties=\@eightiesfrench
1031 \let\@nineties=\@ninetiesfrench
1032 \let\fc@nbrstr@preamble\fc@nbrstr@Fpreamble
1033 \let\fc@nbrstr@postamble\fc@apply@gcase
1034 \@@numberstringfrench{#1}{#2}}
1035 \global\let\@NumberstringFfrenchfrance\@NumberstringFfrenchfrance
1036 \newcommand*{\@NumberstringFfrenchbelgian}[2]{%
1037 \fc@french@common
1038 \let\fc@gcase\fc@UpperCaseFirstLetter
1039 \let\@seventies=\@seventiesfrenchswiss
1040 \let\@eighties=\@eightiesfrench
1041 \let\@nineties=\@ninetiesfrench
1042 \let\fc@nbrstr@preamble\fc@nbrstr@Fpreamble
1043 \let\fc@nbrstr@postamble\fc@apply@gcase
1044 \@@numberstringfrench{#1}{#2}}
1045 \global\let\@NumberstringFfrenchbelgian\@NumberstringFfrenchbelgian
1046 \global\let\@NumberstringFfrench=\@NumberstringFfrenchfrance
1047 \global\let\@NumberstringNfrench\@NumberstringMfrench
1048 \newcommand*{\@ordinalstringMfrenchswiss}[2]{%
1049 \fc@french@common
1050 \let\fc@gcase\fc@CaseIden
1051 \let\fc@first\fc@@firstfrench
1052 \let\@seventies=\@seventiesfrenchswiss
1053 \let\@eighties=\@eightiesfrenchswiss
1054 \let\@nineties=\@ninetiesfrenchswiss
1055 \@@ordinalstringfrench{#1}{#2}%
1056 }%
1057 \global\let\@ordinalstringMfrenchswiss\@ordinalstringMfrenchswiss
1058 \newcommand*\fc@@firstfrench{premier}
1059 \global\let\fc@@firstfrench\fc@@firstfrench

1060 \newcommand*\fc@@firstFfrench{premier\protect\'ere}
1061 \global\let\fc@@firstFfrench\fc@@firstFfrench
1062 \newcommand*{\@ordinalstringMfrenchfrance}[2]{%
1063 \fc@french@common
1064 \let\fc@gcase\fc@CaseIden
1065 \let\fc@first=\fc@@firstfrench
1066 \let\@seventies=\@seventiesfrench
1067 \let\@eighties=\@eightiesfrench
1068 \let\@nineties=\@ninetiesfrench
1069 \@@ordinalstringfrench{#1}{#2}}
1070 \global\let\@ordinalstringMfrenchfrance\@ordinalstringMfrenchfrance
1071 \newcommand*{\@ordinalstringMfrenchbelgian}[2]{%
1072 \fc@french@common
1073 \let\fc@gcase\fc@CaseIden
1074 \let\fc@first=\fc@@firstfrench

```

```

1075 \let\@seventies=\@seventiesfrench
1076 \let\@eighties=\@eightiesfrench
1077 \let\@nineties=\@ninetiesfrench
1078 \@ordinalstringfrench{#1}{#2}%
1079 }%
1080 \global\let\@ordinalstringMfrenchbelgian\@ordinalstringMfrenchbelgian
1081 \global\let\@ordinalstringMfrench=\@ordinalstringMfrenchfrance
1082 \newcommand*\@ordinalstringFfrenchswiss}[2]{%
1083 \fc@french@common
1084 \let\fc@gcase\fc@CaseIden
1085 \let\fc@first\fc@@firstFfrench
1086 \let\@seventies=\@seventiesfrenchswiss
1087 \let\@eighties=\@eightiesfrenchswiss
1088 \let\@nineties=\@ninetiesfrenchswiss
1089 \@ordinalstringfrench{#1}{#2}%
1090 }%
1091 \global\let\@ordinalstringFfrenchswiss\@ordinalstringFfrenchswiss
1092 \newcommand*\@ordinalstringFfrenchfrance}[2]{%
1093 \fc@french@common
1094 \let\fc@gcase\fc@CaseIden
1095 \let\fc@first=\fc@@firstFfrench
1096 \let\@seventies=\@seventiesfrench
1097 \let\@eighties=\@eightiesfrench
1098 \let\@nineties=\@ninetiesfrench
1099 \@ordinalstringfrench{#1}{#2}%
1100 }%
1101 \global\let\@ordinalstringFfrenchfrance\@ordinalstringFfrenchfrance
1102 \newcommand*\@ordinalstringFfrenchbelgian}[2]{%
1103 \fc@french@common
1104 \let\fc@gcase\fc@CaseIden
1105 \let\fc@first=\fc@@firstFfrench
1106 \let\@seventies=\@seventiesfrench
1107 \let\@eighties=\@eightiesfrench
1108 \let\@nineties=\@ninetiesfrench
1109 \@ordinalstringfrench{#1}{#2}%
1110 }%
1111 \global\let\@ordinalstringFfrenchbelgian\@ordinalstringFfrenchbelgian
1112 \global\let\@ordinalstringFfrench=\@ordinalstringFfrenchfrance
1113 \global\let\@ordinalstringNfrench\@ordinalstringMfrench
1114 \newcommand*\@OrdinalstringMfrenchswiss}[2]{%
1115 \fc@french@common
1116 \let\fc@gcase\fc@UpperCaseFirstLetter
1117 \let\fc@first=\fc@@firstfrench
1118 \let\@seventies=\@seventiesfrenchswiss
1119 \let\@eighties=\@eightiesfrenchswiss
1120 \let\@nineties=\@ninetiesfrenchswiss
1121 \@ordinalstringfrench{#1}{#2}%
1122 }%
1123 \global\let\@OrdinalstringMfrenchswiss\@OrdinalstringMfrenchswiss

```

```

1124 \newcommand*{\@OrdinalstringMfrenchfrance}[2]{%
1125 \fc@french@common
1126 \let\fc@gcase\fc@UpperCaseFirstLetter
1127 \let\fc@first\fc@@firstfrench
1128 \let\@seventies=\@seventiesfrench
1129 \let\@eighties=\@eightiesfrench
1130 \let\@nineties=\@ninetiesfrench
1131 \@ordinalstringfrench{#1}{#2}%
1132 }%
1133 \global\let\@OrdinalstringMfrenchfrance\@OrdinalstringMfrenchfrance
1134 \newcommand*{\@OrdinalstringMfrenchbelgian}[2]{%
1135 \fc@french@common
1136 \let\fc@gcase\fc@UpperCaseFirstLetter
1137 \let\fc@first\fc@@firstfrench
1138 \let\@seventies=\@seventiesfrench
1139 \let\@eighties=\@eightiesfrench
1140 \let\@nineties=\@ninetiesfrench
1141 \@ordinalstringfrench{#1}{#2}%
1142 }%
1143 \global\let\@OrdinalstringMfrenchbelgian\@OrdinalstringMfrenchbelgian
1144 \global\let\@OrdinalstringMfrench=\@OrdinalstringMfrenchfrance
1145 \newcommand*{\@OrdinalstringFfrenchswiss}[2]{%
1146 \fc@french@common
1147 \let\fc@gcase\fc@UpperCaseFirstLetter
1148 \let\fc@first\fc@@firstfrench
1149 \let\@seventies=\@seventiesfrenchswiss
1150 \let\@eighties=\@eightiesfrenchswiss
1151 \let\@nineties=\@ninetiesfrenchswiss
1152 \@ordinalstringfrench{#1}{#2}%
1153 }%
1154 \global\let\@OrdinalstringFfrenchswiss\@OrdinalstringFfrenchswiss
1155 \newcommand*{\@OrdinalstringFfrenchfrance}[2]{%
1156 \fc@french@common
1157 \let\fc@gcase\fc@UpperCaseFirstLetter
1158 \let\fc@first\fc@@firstFfrench
1159 \let\@seventies=\@seventiesfrench
1160 \let\@eighties=\@eightiesfrench
1161 \let\@nineties=\@ninetiesfrench
1162 \@ordinalstringfrench{#1}{#2}%
1163 }%
1164 \global\let\@OrdinalstringFfrenchfrance\@OrdinalstringFfrenchfrance
1165 \newcommand*{\@OrdinalstringFfrenchbelgian}[2]{%
1166 \fc@french@common
1167 \let\fc@gcase\fc@UpperCaseFirstLetter
1168 \let\fc@first\fc@@firstFfrench
1169 \let\@seventies=\@seventiesfrench
1170 \let\@eighties=\@eightiesfrench
1171 \let\@nineties=\@ninetiesfrench
1172 \@ordinalstringfrench{#1}{#2}%

```

```

1173}%
1174\global\let\@OrdinalstringFfrenchbelgian\@OrdinalstringFfrenchbelgian
1175\global\let\@OrdinalstringFfrench=\@OrdinalstringFfrenchfrance
1176\global\let\@OrdinalstringNfrench\@OrdinalstringMfrench

```

`\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 vintg, whichever is applicable. First check that the macro is not yet defined.

```

1177\ifcsundef{fc@@do@plural@mark}{}%
1178{\PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1179  '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 vintg, 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.

```

1180\def\fc@@do@plural@mark#1{%
1181  \ifcase\count10 %
1182    #1% 0=always
1183    \or% 1=never
1184    \or% 2=multiple
1185    \ifnum\count1>1 %
1186      #1%
1187    \fi
1188    \or% 3= multiple g-last
1189    \ifnum\count1>1 %
1190      \ifnum\count0=\count6 %
1191        #1%
1192      \fi
1193    \fi
1194    \or% 4= multiple l-last
1195    \ifnum\count1>1 %
1196      \ifnum\count9=1 %
1197        \else
1198          #1%
1199        \fi
1200    \fi
1201    \or% 5= multiple lng-last
1202    \ifnum\count1>1 %
1203      \ifnum\count9=1 %
1204        \else
1205          \if\count0>\count6 %
1206            #1%
1207          \fi

```

```

1208     \fi
1209     \fi
1210 \or% 6= multiple ng-last
1211     \ifnum\count1>1 %
1212         \ifnum\count0>\count6 %
1213             #1%
1214         \fi
1215     \fi
1216 \fi
1217}%
1218\global\let\fc@@do@plural@mark\fc@@do@plural@mark
\@nabrstr@FpreambleMacro \fc@@nabrstr@Fpreamble do the necessary preliminaries before formatting a cardinal
with feminine gender.
1219\ifcsundef\fc@@nabrstr@Fpreamble\{}{%
1220 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1221     'fc@@nabrstr@Fpreamble'}}
\@nabrstr@Fpreamble
1222\def\fc@@nabrstr@Fpreamble{%
1223 \fc@read@unit{\count1}{0}%
1224 \ifnum\count1=1 %
1225     \let\fc@wcase@save\fc@wcase
1226     \def\fc@wcase{\noexpand\fc@wcase}%
1227     \def\@nil{\noexpand\@nil}%
1228     \let\fc@nabrstr@postamble\fc@@nabrstr@Fpostamble
1229 \fi
1230}%
1231\global\let\fc@@nabrstr@Fpreamble\fc@@nabrstr@Fpreamble
\@nabrstr@Fpostamble
1232\def\fc@@nabrstr@Fpostamble{%
1233 \let\fc@wcase\fc@wcase@save
1234 \expandafter\fc@get@last@word\expandafter{\@tempa}\@tempb\@tempc
1235 \def\@tempd{un}%
1236 \ifx\@tempc\@tempd
1237     \let\@tempc\@tempa
1238     \edef\@tempa{\@tempb\fc@wcase une\@nil}%
1239 \fi
1240}%
1241\global\let\fc@@nabrstr@Fpostamble\fc@@nabrstr@Fpostamble
\@pot@longscalefrenchMacro \fc@@pot@longscalefrench is used to produce powers of ten with long scale con-
vention. The long scale convention is correct for French and elsewhere in Europe. First we
check that the macro is not yet defined.
1242\ifcsundef\fc@@pot@longscalefrench\{}{%
1243 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1244     'fc@@pot@longscalefrench'}}
Argument are as follows:

```

#1 input, plural value of d , that is to say: let d be the number multiplying the considered power of ten, then the plural value #2 is expected to be 0 if $d = 0$, 1 if $d = 1$, or > 1 if $d > 1$

#2 output, counter, maybe 0 when power of ten is 1, 1 when power of ten starts with “mil(le)”, or 2 when power of ten is a “ $\langle n \rangle$ illion(s)| $\langle n \rangle$ illiard(s)”

#3 output, macro into which to place the formatted power of ten

Implicit arguments as follows:

`\count0` input, counter giving the weight w , this is expected to be multiple of 3

```

1245 \def\fc@pot@longscalefrench#1#2#3{%
1246   {%

```

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

```

1247   \edef\@tempb{\number#1}%

```

Let `\count1` be the plural value.

```

1248   \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 .

```

1249   \count2\count0 %
1250   \divide\count2 by 6 %
1251   \count3\count2 %
1252   \multiply\count3 by 6 %
1253   \count3-\count3 %
1254   \advance\count3 by \count0 %
1255   \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)”.

```

1256   \ifnum\count1>0 %

```

Plural value is > 0 so have at least one “mil(le)” or “ $\langle n \rangle$ illion(s)” or “ $\langle n \rangle$ illiard(s)”. We need to distinguish between the case of “mil(le)” and that of “ $\langle n \rangle$ illion(s)” or “ $\langle n \rangle$ illiard(s)”, so we `\define \@temp` to ‘1’ for “mil(le)”, and to ‘2’ otherwise.

```

1257   \edef\@temp{%
1258     \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.

```

1259       1%
1260     \else
1261       \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`.

```

1262       \ifnum\fc@longscale@nilliard@upto=0 %

```

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

```

1263       2%
1264     \else

```

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

```

1265         \ifnum\count2>\fc@longscale@nilliard@upto
1266             1%
1267         \else
1268             2%
1269         \fi
1270     \fi
1271 \else
1272     2%
1273 \fi
1274 \fi
1275 }%
1276 \ifnum\@temph=1 %

```

Here 10^w is formatted as “mil(le)”.

```

1277     \count10=\fc@frenchoptions@mil@plural\space
1278     \edef\@tempe{%
1279         \noexpand\fc@wcase
1280         mil%
1281         \fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark
1282         \noexpand\@nil
1283     }%
1284 \else
1285     % weight >= 6
1286     \expandafter\fc@@latin@cardinal@prefix\expandafter{\the\count2}\@tempg
1287     % now form the xxx-illion(s) or xxx-illiard(s) word
1288     \ifnum\count3>2 %
1289         \toks10{illiard}%
1290         \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space
1291     \else
1292         \toks10{illion}%
1293         \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
1294     \fi
1295     \edef\@tempe{%
1296         \noexpand\fc@wcase
1297         \@tempg
1298         \the\toks10 %
1299         \fc@@do@plural@mark s%
1300         \noexpand\@nil
1301     }%
1302 \fi
1303 \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.

```

1304     \let\@tempe\@empty
1305     \def\@temph{0}%
1306 \fi
1307 \else

```


Case of $w = 0$.

```
1308 \let\@tempe\@empty
1309 \def\@temph{0}%
1310 \fi
```

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

```
1311 \expandafter\toks\expandafter1\expandafter{\@tempe}%
1312 \toks0{#2}%
1313 \edef\@tempa{\the\toks0 \@temph \def\noexpand#3{\the\toks1}}%
1314 \expandafter
1315 }\@tempa
1316}%
1317\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.

```
1318\ifcsundef{fc@@pot@shortscalefrench}{\}%
1319 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1320 '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

```
1321\def\fc@@pot@shortscalefrench#1#2#3{%
1322 {%
```

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

```
1323 \edef\@tempb{\number#1}%
```

And let `\count1` be the plural value.

```
1324 \count1=\@tempb
```

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

```
1325 \count2\count0 %
1326 \divide\count2 by 3 %
1327 \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)”.

```
1328 \ifnum\count0>0 % If weight>=3, i.e we do have to append thousand or n-illion(s)/n-illiard(s)
```

```

1329 \ifnum\count1>0 % we have at least one thousand/n-illion/n-illiard
1330 \ifnum\count2=0 %
1331 \def\@temph{1}%
1332 \count1=\fc@frenchoptions@mil@plural\space
1333 \edef\@tempe{%
1334 mil%
1335 \fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark
1336 }%
1337 \else
1338 \def\@temph{2}%
1339 % weight >= 6
1340 \expandafter\fc@@latin@cardinal@prefix\expandafter{\the\count2}\@tempg
1341 \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
1342 \edef\@tempe{%
1343 \noexpand\fc@wcase
1344 \@tempg
1345 illion%
1346 \fc@@do@plural@mark s%
1347 \noexpand\@nil
1348 }%
1349 \fi
1350 \else

```

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

```

1351 \def\@temph{0}%
1352 \let\@tempe\@empty
1353 \fi
1354 \else

```

Here $w = 0$.

```

1355 \def\@temph{0}%
1356 \let\@tempe\@empty
1357 \fi
1358 % now place into \cs{@tempa} the assignment of results \cs{@temph} and \cs{@tempe} to to \text
1359 % \texttt{\#3} for further propagation after closing brace.
1360 % \begin{macrocode}
1361 \expandafter\toks\expandafter1\expandafter{\@tempe}%
1362 \toks0{\#2}%
1363 \edef\@tempa{\the\toks0 \@temph \def\noexpand\#3{\the\toks1}}%
1364 \expandafter
1365 }\@tempa
1366}%
1367\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.

```

1368\ifcsundef{fc@@pot@recursivefrench}{\%
1369 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1370 '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

```
1371 \def\fc@pot@recursivefrench#1#2#3{%
1372   {%
```

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

```
1373   \edef\@tempb{\number#1}%
1374   \let\@tempa\@tempa
```

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

```
1375   \count1=\@tempb\space
```

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

```
1376   \ifnum\count1>0 %
1377     \count2\count0 %
1378     \divide\count2 by 9 %
1379     \advance\count2 by -1 %
1380     \let\@tempe\@empty
1381     \edef\@tempf{\fc@frenchoptions@supermillion@dos
1382       de\fc@frenchoptions@supermillion@dos\fc@wcase milliards\@nil}%
1383     \count11\count0 %
1384     \ifnum\count2>0 %
1385       \count3\count2 %
1386       \count3-\count3 %
1387       \multiply\count3 by 9 %
1388       \advance\count11 by \count3 %
1389       \loop
1390         % (\count2, \count3) <- (\count2 div 2, \count2 mod 2)
1391         \count3\count2 %
1392         \divide\count3 by 2 %
1393         \multiply\count3 by 2 %
1394         \count3-\count3 %
1395         \advance\count3 by \count2 %
1396         \divide\count2 by 2 %
1397         \ifnum\count3=1 %
1398           \let\@tempg\@tempe
1399           \edef\@tempe{\@tempg\@tempf}%
1400         \fi
1401         \let\@tempg\@tempf
1402         \edef\@tempf{\@tempg\@tempg}%
1403       \ifnum\count2>0 %
1404     \repeat
1405   \fi
1406   \divide\count11 by 3 %
```

```

1407 \ifcase\count11 % 0 .. 5
1408 % 0 => d milliard(s) (de milliards)*
1409 \def\@temph{2}%
1410 \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space
1411 \or % 1 => d mille milliard(s) (de milliards)*
1412 \def\@temph{1}%
1413 \count10=\fc@frenchoptions@mil@plural\space
1414 \or % 2 => d million(s) (de milliards)*
1415 \def\@temph{2}%
1416 \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
1417 \or % 3 => d milliard(s) (de milliards)*
1418 \def\@temph{2}%
1419 \count10=\csname fc@frenchoptions@n-illiard@plural\endcsname\space
1420 \or % 4 => d mille milliards (de milliards)*
1421 \def\@temph{1}%
1422 \count10=\fc@frenchoptions@mil@plural\space
1423 \else % 5 => d million(s) (de milliards)*
1424 \def\@temph{2}%
1425 \count10=\csname fc@frenchoptions@n-illion@plural\endcsname\space
1426 \fi
1427 \let\@tempg\@tempe
1428 \edef\@tempf{%
1429 \ifcase\count11 % 0 .. 5
1430 \or
1431 mil\fc@@do@plural@mark \fc@frenchoptions@mil@plural@mark
1432 \or
1433 million\fc@@do@plural@mark s%
1434 \or
1435 milliard\fc@@do@plural@mark s%
1436 \or
1437 mil\fc@@do@plural@mark\fc@frenchoptions@mil@plural@mark
1438 \noexpand\@nil\fc@frenchoptions@supermillion@dos
1439 \noexpand\fc@wcase milliards% 4
1440 \or
1441 million\fc@@do@plural@mark s%
1442 \noexpand\@nil\fc@frenchoptions@supermillion@dos
1443 de\fc@frenchoptions@supermillion@dos\noexpand\fc@wcase milliards% 5
1444 \fi
1445 }%
1446 \edef\@tempe{%
1447 \ifx\@tempf\@empty\else
1448 \expandafter\fc@wcase\@tempf\@nil
1449 \fi
1450 \@tempg
1451 }%
1452 \else
1453 \def\@temph{0}%
1454 \let\@tempe\@empty
1455 \fi

```

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

```

1456 \expandafter\toks\expandafter1\expandafter{\@tempe}%
1457 \toks0{#2}%
1458 \edef\@tempa{\the\toks0 \@temph \def\noexpand#3{\the\toks1}}%
1459 \expandafter
1460 }\@tempa
1461 }%
1462 \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.

```

1463 \ifcsundef{fc@muladdfrench}{}%
1464 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1465 '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

```

1466 \def\fc@muladdfrench#1#2#3{%
1467   {%

```

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

```

1468 \edef\@tempc{#1}%
1469 \edef\@tempd{#2}%
1470 \edef\@tempf{#3}%
1471 \let\@tempc\@tempc
1472 \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”.

```

1473 \ifnum\@tempc=1 %
1474 \ifnum\count9=1 %
1475 \let\@tempd\empty
1476 \fi
1477 \fi

```

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

```

1478 \edef\@tempg{%
1479 \@tempd
1480 \ifx\@tempd\@empty\else
1481 \ifx\@tempf\@empty\else
1482 \ifcase\count9 %
1483 \or
1484 \fc@frenchoptions@submillion@dos
1485 \or
1486 \fc@frenchoptions@supermillion@dos
1487 \fi
1488 \fi
1489 \fi
1490 \@tempf
1491 }%

```

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

```

1492 \edef\@temph{%
1493 \@tempa
1494 \ifx\@tempa\@empty\else
1495 \ifx\@tempg\@empty\else
1496 \ifcase\count8 %
1497 \or
1498 \fc@frenchoptions@submillion@dos
1499 \or
1500 \fc@frenchoptions@supermillion@dos
1501 \fi
1502 \fi
1503 \fi
1504 \@tempg
1505 }%

```

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

```

1506 \def\@tempb##1{\def\@tempa{\def\@tempa{##1}}}%
1507 \expandafter\@tempb\expandafter{\@temph}%
1508 \expandafter
1509 }\@tempa
1510 }%
1511 \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.

```

1512 \ifcsundef{fc@lthundredstringfrench}{\}%
1513 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1514 'fc@lthundredstringfrench'}}

```

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

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

#1 input/output macro to which append the result

Implicit input arguments as follows:

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

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

```
1515 \def\fc@lthundredstringfrench#1{%
```

```
1516   {%
```

First save arguments into local temporary macro.

```
1517   \let\@tempc#1%
```

Read units d_w to `\count1`.

```
1518   \fc@read@unit{\count1}{\count0}%
```

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

```
1519   \count3\count0 %
```

```
1520   \advance\count3 1 %
```

```
1521   \fc@read@unit{\count2}{\count3}%
```

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

```
1522   \edef\@tempa{%
```

```
1523     \@tempc
```

```
1524     \ifnum\count2>1 %
```

```
1525       % 20 .. 99
```

```
1526       \ifnum\count2>6 %
```

```
1527         % 70 .. 99
```

```
1528         \ifnum\count2<8 %
```

```
1529           % 70 .. 79
```

```
1530           \@seventies{\count1}%
```

```
1531         \else
```

```
1532           % 80..99
```

```
1533           \ifnum\count2<9 %
```

```
1534             % 80 .. 89
```

```
1535             \@eighties{\count1}%
```

```
1536           \else
```

```
1537             % 90 .. 99
```

```
1538             \@nineties{\count1}%
```

```
1539           \fi
```

```
1540         \fi
```

```
1541       \else
```

```
1542         % 20..69
```

```
1543         \@tenstring{\count2}%
```

```
1544         \ifnum\count1>0 %
```

```
1545           % x1 .. x0
```

```
1546           \ifnum\count1=1 %
```

```
1547             % x1
```

```
1548             \fc@frenchoptions@submillion@dos\@andname\fc@frenchoptions@submillion@dos
```

```
1549           \else
```

```
1550             % x2 .. x9
```

```

1551         -%
1552         \fi
1553         \@unitstring{\count1}%
1554     \fi
1555     \fi
1556 \else
1557     % 0 .. 19
1558     \ifnum\count2=0 % when tens = 0
1559     % 0 .. 9
1560     \ifnum\count1=0 % when units = 0
1561     % \count3=1 when #1 = 0, i.e. only for the unit of the top level number
1562     \ifnum\count3=1 %
1563     \ifnum\fc@max@weight=0 %
1564     \@unitstring{0}%
1565     \fi
1566     \fi
1567 \else
1568     % 1 .. 9
1569     \@unitstring{\count1}%
1570 \fi
1571 \else
1572     % 10 .. 19
1573     \@teenstring{\count1}%
1574 \fi
1575 \fi
1576 }%

```

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

```

1577 \def\@tempb##1{\def\@tempa{\def#1{##1}}}%
1578 \expandafter\@tempb\expandafter{\@tempa}%
1579 \expandafter
1580 }\@tempa
1581 }%
1582 \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.

```

1583 \ifcsundef{fc@ltthousandstringfrench}{\}%
1584 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1585   '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)

```

1586 \def\fc@ltthousandstringfrench#1{%
1587   {%

```


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

```
1588 \count4\count0 %
1589 \advance\count4 by 2 %
1590 \fc@read@unit{\count2 }{\count4 }%
```

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

```
1591 \advance\count4 by -1 %
1592 \count3\count4 %
1593 \advance\count3 by -1 %
1594 \fc@check@nonzeros{\count3 }{\count4 }\@tempa
```

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

```
1595 \edef\@temps{%
1596 \ifcase\fc@frenchoptions@cent@plural\space
1597 % 0 => always
1598 s%
1599 \or
1600 % 1 => never
1601 \or
1602 % 2 => multiple
1603 \ifnum\count2>1s\fi
1604 \or
1605 % 3 => multiple g-last
1606 \ifnum\count2>1 \ifnum\@tempa=0 \ifnum\count0=\count6s\fi\fi\fi
1607 \or
1608 % 4 => multiple l-last
1609 \ifnum\count2>1 \ifnum\@tempa=0 \ifnum\count9=0s\else\ifnum\count9=2s\fi\fi\fi\fi
1610 \fi
1611 }%
1612 % compute spacing after cent(s?) into \@tempb
1613 \expandafter\let\expandafter\@tempb
1614 \ifnum\@tempa>0 \fc@frenchoptions@submillion@dos\else\@empty\fi
1615 % now place into \@tempa the hundreds
1616 \edef\@tempa{%
1617 \ifnum\count2=0 %
1618 \else
1619 \ifnum\count2=1 %
1620 \expandafter\fc@wcase\@hundred\@nil
1621 \else
1622 \@unitstring{\count2}\fc@frenchoptions@submillion@dos
1623 \noexpand\fc@wcase\@hundred\@temps\noexpand\@nil
1624 \fi
1625 \@tempb
1626 \fi
1627 }%
1628 % now append to \@tempa the ten and unit
1629 \fc@lthundredstringfrench\@tempa
```

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

```
1630 \def\@tempb##1{\def\@tempa{\def#1{##1}}}%

```

```

1631 \expandafter\@tempb\expandafter{\@tempa}%
1632 \expandafter
1633 }\@tempa
1634}%
1635\global\let\fc@ltthousandstringfrench\fc@ltthousandstringfrench
numberstringfrenchMacro \@@numberstringfrench is the main engine for formatting cadinal numbers in
French. First we check that the control sequence is not yet defined.
1636\ifcsundef{\@@numberstringfrench}\{%
1637 \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
1638\def\@@numberstringfrench#1#2{%
1639 {%
First parse input number to be formatted and do some error handling.
1640 \edef\@tempa{#1}%
1641 \expandafter\fc@number@parser\expandafter{\@tempa}%
1642 \ifnum\fc@min@weight<0 %
1643 \PackageError{fmtcount}{Out of range}%
1644 {This macro does not work with fractional numbers}%
1645 \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’.
1646 \edef\@tempa{\ifcase\fc@sign@case\space\or\fc@wcase plus\@nil\or\fc@wcase moins\@nil\fi}%
1647 \fc@nbrstr@preamble
1648 \fc@@nbrstrfrench@inner
1649 \fc@nbrstr@postamble
Propagate the result — i.e. expansion of \@tempa — into macro #2 after closing brace.
1650 \def\@tempb##1{\def\@tempa{\def#2{##1}}}%
1651 \expandafter\@tempb\expandafter{\@tempa}%
1652 \expandafter
1653 }\@tempa
1654}%
1655\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.
1656\def\fc@@nbrstrfrench@inner{%
Now loop, first we compute starting weight as  $3 \times \left\lfloor \frac{\fc@max@weight}{3} \right\rfloor$  into \count0.
1657 \count0=\fc@max@weight
1658 \divide\count0 by 3 %
1659 \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`.

```
1660 \fc@intpart@find@last{\count5 }%
1661 \count6\count5 %
1662 \divide\count6 3 %
1663 \multiply\count6 3 %
1664 \count8=0 %
1665 \loop
```

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

```
1666 \count1\count0 %
1667 \advance\count1 by 2 %
1668 \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`.

```
1669 \fc@poweroften\@tempt{\count9 }\@tempb
```

Now we generate the formatted number d into macro `\@tempd` by which we need to multiply 10^w . Implicit input argument is `\count9` for power type of 10^9 , and `\count6`

```
1670 \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 .

```
1671 \fc@muladdfrench\@tempt\@tempd\@tempb
```

Then iterate.

```
1672 \count8\count9 %
1673 \advance\count0 by -3 %
1674 \ifnum\count6>\count0 \else
1675 \repeat
1676}%
1677\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.

```
1678\ifcsundef{\@@ordinalstringfrench}{\%
1679 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro
1680 '@@ordinalstringfrench'}}}
```

Arguments are as follows:

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

```
1681\def\@@ordinalstringfrench#1#2{%
1682 {%
```

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

```
1683 \edef\@tempa{#1}%
1684 \expandafter\fc@number@parser\expandafter{\@tempa}%
```

```

1685 \ifnum\fc@min@weight<0 %
1686   \PackageError{fmtcount}{Out of range}%
1687   {This macro does not work with fractional numbers}%
1688 \fi
1689 \ifnum\fc@sign@case>0 %
1690   \PackageError{fmtcount}{Out of range}%
1691   {This macro does with negative or explicitly marked as positive numbers}%
1692 \fi

```

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

```

1693 \ifnum\fc@max@weight=0 %
1694   \ifnum\csname fc@digit@0\endcsname=1 %
1695     \count0=1 %
1696   \else
1697     \count0=0 %
1698   \fi
1699 \else
1700   \count0=0 %
1701 \fi
1702 \ifnum\count0=1 %
1703   \protected@edef\@tempa{\expandafter\fc@wcase\fc@first\@nil}%
1704 \else

```

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

```

1705 \def\@tempa##1{%
1706   \expandafter\edef\csname fc@frenchoptions@##1@plural\endcsname{%
1707     \ifcase\csname fc@frenchoptions@##1@plural\endcsname\space
1708       0% 0: always => always
1709       \or
1710       1% 1: never => never
1711       \or
1712       6% 2: multiple => multiple ng-last
1713       \or
1714       1% 3: multiple g-last => never
1715       \or
1716       5% 4: multiple l-last => multiple lng-last
1717       \or
1718       5% 5: multiple lng-last => multiple lng-last
1719       \or
1720       6% 6: multiple ng-last => multiple ng-last
1721     \fi
1722   }%
1723 }%
1724 \@tempa{vingt}%
1725 \@tempa{cent}%
1726 \@tempa{mil}%
1727 \@tempa{n-illion}%

```

```

1728      \@tempa{n-illiard}%
Now make \fc@wcase and \@nil non expandable
1729      \let\fc@wcase@save\fc@wcase
1730      \def\fc@wcase{\noexpand\fc@wcase}%
1731      \def\@nil{\noexpand\@nil}%

In the sequel, \@tempa is used to accumulate the formatted number.
1732      \let\@tempa\@empty
1733      \fc@@nbrstrfrench@inner

Now restore \fc@wcase
1734      \let\fc@wcase\fc@wcase@save

Now we add the “ième” ending
1735      \expandafter\fc@get@last@word\expandafter{\@tempa}\@tempb\@tempc
1736      \expandafter\fc@get@last@letter\expandafter{\@tempc}\@tempd\@tempe
1737      \def\@tempf{e}%
1738      \ifx\@tempe\@tempf
1739          \protected@edef\@tempa{\@tempb\expandafter\fc@wcase\@tempd i\protect\'eme\@nil}%
1740      \else
1741          \def\@tempf{q}%
1742          \ifx\@tempe\@tempf
1743              \protected@edef\@tempa{\@tempb\expandafter\fc@wcase\@tempd qui\protect\'eme\@nil}%
1744          \else
1745              \def\@tempf{f}%
1746              \ifx\@tempe\@tempf
1747                  \protected@edef\@tempa{\@tempb\expandafter\fc@wcase\@tempd vi\protect\'eme\@nil}%
1748              \else
1749                  \protected@edef\@tempa{\@tempb\expandafter\fc@wcase\@tempc i\protect\'eme\@nil}%
1750              \fi
1751          \fi
1752      \fi
1753  \fi

Apply \fc@gc case to the result.
1754      \fc@apply@gc case

Propagate the result — i.e. expansion of \@tempa — into macro #2 after closing brace.
1755      \def\@tempb##1{\def\@tempa{\def#2{##1}}}%
1756      \expandafter\@tempb\expandafter{\@tempa}%
1757      \expandafter
1758  }\@tempa
1759}%

1760\global\let\@@ordinalstringfrench\@@ordinalstringfrench

Macro \fc@frenchoptions@setdefaults allows to set all options to default for the French.
1761\newcommand*\fc@frenchoptions@setdefaults{%
1762  \csname KV@fcfrench@all plural\endcsname{reformed}%
1763  \fc@gl@def\fc@frenchoptions@submillion@dos{-}%
1764  \fc@gl@let\fc@frenchoptions@supermillion@dos\space
1765  \fc@gl@let\fc@u@in@duo\@empty% Could be ‘u’

```

```

1766 % \fc@gl@let\fc@poweroften\fc@pot@longscalefrench
1767 \fc@gl@let\fc@poweroften\fc@pot@recursivefrench
1768 \fc@gl@def\fc@longscale@illiard@upto{0}% infinity
1769 \fc@gl@def\fc@frenchoptions@mil@plural@mark{le}%
1770}%
1771\global\let\fc@frenchoptions@setdefaults\fc@frenchoptions@setdefaults
1772{%
1773 \let\fc@gl@def\gdef
1774 \def\fc@gl@let{\global\let}%
1775 \fc@frenchoptions@setdefaults
1776}%

```

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

```

1777\gdef\@ordinalstringMfrench{\csuse{\ordinalstringMfrench\fmtcount@french}}%
1778\gdef\@ordinalstringFfrench{\csuse{\ordinalstringFfrench\fmtcount@french}}%
1779\gdef\@OrdinalstringMfrench{\csuse{\OrdinalstringMfrench\fmtcount@french}}%
1780\gdef\@OrdinalstringFfrench{\csuse{\OrdinalstringFfrench\fmtcount@french}}%
1781\gdef\@numberstringMfrench{\csuse{\numberstringMfrench\fmtcount@french}}%
1782\gdef\@numberstringFfrench{\csuse{\numberstringFfrench\fmtcount@french}}%
1783\gdef\@NumberstringMfrench{\csuse{\NumberstringMfrench\fmtcount@french}}%
1784\gdef\@NumberstringFfrench{\csuse{\NumberstringFfrench\fmtcount@french}}%

```

10.1.7 fc-frenchb.def

```

1785\ProvidesFCLanguage{frenchb}[2013/08/17]%
1786\FCloadlang{french}%

```

Set frenchb to be equivalent to french.

```

1787\global\let\@ordinalMfrenchb=\@ordinalMfrench
1788\global\let\@ordinalFfrenchb=\@ordinalFfrench
1789\global\let\@ordinalNfrenchb=\@ordinalNfrench
1790\global\let\@numberstringMfrenchb=\@numberstringMfrench
1791\global\let\@numberstringFfrenchb=\@numberstringFfrench
1792\global\let\@numberstringNfrenchb=\@numberstringNfrench
1793\global\let\@NumberstringMfrenchb=\@NumberstringMfrench
1794\global\let\@NumberstringFfrenchb=\@NumberstringFfrench
1795\global\let\@NumberstringNfrenchb=\@NumberstringNfrench
1796\global\let\@ordinalstringMfrenchb=\@ordinalstringMfrench
1797\global\let\@ordinalstringFfrenchb=\@ordinalstringFfrench
1798\global\let\@ordinalstringNfrenchb=\@ordinalstringNfrench
1799\global\let\@OrdinalstringMfrenchb=\@OrdinalstringMfrench
1800\global\let\@OrdinalstringFfrenchb=\@OrdinalstringFfrench
1801\global\let\@OrdinalstringNfrenchb=\@OrdinalstringNfrench

```

10.1.8 fc-german.def

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

```

1802\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:

```

1803 \newcommand{\@ordinalMgerman}[2]{%
1804   \edef#2{\number#1\relax.}%
1805 }%
1806 \global\let\@ordinalMgerman\@ordinalMgerman

```

Feminine:

```

1807 \newcommand{\@ordinalFgerman}[2]{%
1808   \edef#2{\number#1\relax.}%
1809 }%
1810 \global\let\@ordinalFgerman\@ordinalFgerman

```

Neuter:

```

1811 \newcommand{\@ordinalNgerman}[2]{%
1812   \edef#2{\number#1\relax.}%
1813 }%
1814 \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):

```

1815 \newcommand*{\@unitstringgerman}[1]{%
1816   \ifcase#1%
1817     null%
1818     \or ein%
1819     \or zwei%
1820     \or drei%
1821     \or vier%
1822     \or f"unf%
1823     \or sechs%
1824     \or sieben%
1825     \or acht%
1826     \or neun%
1827   \fi
1828 }%
1829 \global\let\@unitstringgerman\@unitstringgerman

```

Tens (argument must go from 1 to 10):

```

1830 \newcommand*{\@tenstringgerman}[1]{%
1831   \ifcase#1%
1832     \or zehn%
1833     \or zwanzig%
1834     \or drei{\ss}ig%
1835     \or vierzig%
1836     \or f"unfzig%
1837     \or sechzig%
1838     \or siebzig%
1839     \or achtzig%
1840     \or neunzig%
1841     \or einhundert%
1842   \fi
1843 }%
1844 \global\let\@tenstringgerman\@tenstringgerman

```

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

```
1845 \providecommand*\einhundert{\einhundert}%
1846 \providecommand*\eintausend{\eintausend}%
1847 \global\let\einhundert\einhundert
1848 \global\let\eintausend\eintausend
```

Teens:

```
1849 \newcommand*{\@@teenstringgerman[1]{%
1850   \ifcase#1%
1851     zehn%
1852     \or elf%
1853     \or zw"olf%
1854     \or dreizehn%
1855     \or vierzehn%
1856     \or f"unfzehn%
1857     \or sechzehn%
1858     \or siebzehn%
1859     \or achtzehn%
1860     \or neunzehn%
1861   \fi
1862 }%
1863 \global\let\@@teenstringgerman\@@teenstringgerman
```

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

```
1864 \newcommand*{\@numberstringMgerman}[2]{%
1865   \let\@unitstring=\@unitstringgerman
1866   \let\@teenstring=\@teenstringgerman
1867   \let\@tenstring=\@tenstringgerman
1868   \@numberstringgerman{#1}{#2}%
1869 }%
1870 \global\let\@numberstringMgerman\@numberstringMgerman
```

Feminine and neuter forms:

```
1871 \global\let\@numberstringFgerman=\@numberstringMgerman
1872 \global\let\@numberstringNgerman=\@numberstringMgerman
```

As above, but initial letters in upper case:

```
1873 \newcommand*{\@NumberstringMgerman}[2]{%
1874   \@numberstringMgerman{#1}{\@num@str}%
1875   \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
1876 }%
1877 \global\let\@NumberstringMgerman\@NumberstringMgerman
```

Feminine and neuter form:

```
1878 \global\let\@NumberstringFgerman=\@NumberstringMgerman
1879 \global\let\@NumberstringNgerman=\@NumberstringMgerman
```

As above, but for ordinals.

```
1880 \newcommand*{\@ordinalstringMgerman}[2]{%
1881   \let\@unitthstring=\@unitthstringgerman
```



```

1882 \let\@teenthstring=\@teenthstringMgerman
1883 \let\@tenthstring=\@tenthstringMgerman
1884 \let\@unitstring=\@unitstringgerman
1885 \let\@teenstring=\@teenstringgerman
1886 \let\@tenstring=\@tenstringgerman
1887 \def\@thousandth{tausendster}%
1888 \def\@hundredth{hundertster}%
1889 \@ordinalstringgerman{#1}{#2}%
1890 }%
1891 \global\let\@ordinalstringMgerman\@ordinalstringMgerman

```

Feminine form:

```

1892 \newcommand*\@ordinalstringFgerman}[2]{%
1893   \let\@unitthstring=\@unitthstringFgerman
1894   \let\@teenthstring=\@teenthstringFgerman
1895   \let\@tenthstring=\@tenthstringFgerman
1896   \let\@unitstring=\@unitstringgerman
1897   \let\@teenstring=\@teenstringgerman
1898   \let\@tenstring=\@tenstringgerman
1899   \def\@thousandth{tausendste}%
1900   \def\@hundredth{hundertste}%
1901   \@ordinalstringgerman{#1}{#2}%
1902 }%
1903 \global\let\@ordinalstringFgerman\@ordinalstringFgerman

```

Neuter form:

```

1904 \newcommand*\@ordinalstringNgerman}[2]{%
1905   \let\@unitthstring=\@unitthstringNgerman
1906   \let\@teenthstring=\@teenthstringNgerman
1907   \let\@tenthstring=\@tenthstringNgerman
1908   \let\@unitstring=\@unitstringgerman
1909   \let\@teenstring=\@teenstringgerman
1910   \let\@tenstring=\@tenstringgerman
1911   \def\@thousandth{tausendstes}%
1912   \def\@hundredth{hunderstes}%
1913   \@ordinalstringgerman{#1}{#2}%
1914 }%
1915 \global\let\@ordinalstringNgerman\@ordinalstringNgerman

```

As above, but with initial letters in upper case.

```

1916 \newcommand*\@OrdinalstringMgerman}[2]{%
1917   \@ordinalstringMgerman{#1}{\@num@str}%
1918   \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
1919 }%
1920 \global\let\@OrdinalstringMgerman\@OrdinalstringMgerman

```

Feminine form:

```

1921 \newcommand*\@OrdinalstringFgerman}[2]{%
1922   \@ordinalstringFgerman{#1}{\@num@str}%
1923   \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
1924 }%

```

```
1925 \global\let\@OrdinalstringFgerman\@OrdinalstringFgerman
```

Neuter form:

```
1926 \newcommand*{\@OrdinalstringNgerman}[2]{%
1927 \@OrdinalstringNgerman{#1}{\@num@str}%
1928 \edef#2{\noexpand\MakeUppercase\expandonce\@num@str}%
1929 }%
1930 \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:

```
1931 \newcommand*\@unitthstringMgerman[1]{%
1932 \ifcase#1%
1933 nullter%
1934 \or erster%
1935 \or zweiter%
1936 \or dritter%
1937 \or vierter%
1938 \or f\"unfter%
1939 \or sechster%
1940 \or siebter%
1941 \or achter%
1942 \or neunter%
1943 \fi
1944 }%
1945 \global\let\@unitthstringMgerman\@unitthstringMgerman
```

Tens:

```
1946 \newcommand*\@tenthstringMgerman[1]{%
1947 \ifcase#1%
1948 \or zehnter%
1949 \or zwanzigster%
1950 \or drei{\ss}igster%
1951 \or vierzigster%
1952 \or f\"unzigster%
1953 \or sechzigster%
1954 \or siebzigster%
1955 \or achtzigster%
1956 \or neunzigster%
1957 \fi
1958 }%
1959 \global\let\@tenthstringMgerman\@tenthstringMgerman
```

Teens:

```
1960 \newcommand*\@teenthstringMgerman[1]{%
1961 \ifcase#1%
1962 zehnter%
1963 \or elfter%
1964 \or zw\"olfter%
1965 \or dreizehnter%
1966 \or vierzehnter%
```

```

1967 \or f\"unfzehnter%
1968 \or sechzehnter%
1969 \or siebzehnter%
1970 \or achtzehnter%
1971 \or neunzehnter%
1972 \fi
1973 }%
1974 \global\let\@@teenthstringMgerman\@@teenthstringMgerman

```

Units (feminine):

```

1975 \newcommand*\@@unitthstringFgerman[1]{%
1976 \ifcase#1%
1977 nullte%
1978 \or erste%
1979 \or zweite%
1980 \or dritte%
1981 \or vierte%
1982 \or f\"unfte%
1983 \or sechste%
1984 \or siebte%
1985 \or achte%
1986 \or neunte%
1987 \fi
1988 }%
1989 \global\let\@@unitthstringFgerman\@@unitthstringFgerman

```

Tens (feminine):

```

1990 \newcommand*\@@tenthstringFgerman[1]{%
1991 \ifcase#1%
1992 \or zehnte%
1993 \or zwanzigste%
1994 \or drei{\ss}igste%
1995 \or vierzigste%
1996 \or f\"unfzigste%
1997 \or sechzigste%
1998 \or siebzigste%
1999 \or achtzigste%
2000 \or neunzigste%
2001 \fi
2002 }%
2003 \global\let\@@tenthstringFgerman\@@tenthstringFgerman

```

Teens (feminine)

```

2004 \newcommand*\@@teenthstringFgerman[1]{%
2005 \ifcase#1%
2006 zehnte%
2007 \or elfte%
2008 \or zw\"olfte%
2009 \or dreizehnte%
2010 \or vierzehnte%
2011 \or f\"unfzehnte%

```

```

2012    \or sechzehnte%
2013    \or siebzehnte%
2014    \or achtzehnte%
2015    \or neunzehnte%
2016    \fi
2017}%
2018\global\let\@@teenthstringFgerman\@@teenthstringFgerman

    Units (neuter):
2019\newcommand*\@@unitthstringNgerman[1]{%
2020    \ifcase#1%
2021        nulltes%
2022        \or erstes%
2023        \or zweites%
2024        \or drittes%
2025        \or viertes%
2026        \or f\"unftes%
2027        \or sechstes%
2028        \or siebtes%
2029        \or achttes%
2030        \or neuntes%
2031    \fi
2032}%
2033\global\let\@@unitthstringNgerman\@@unitthstringNgerman

    Tens (neuter):
2034\newcommand*\@@tenthstringNgerman[1]{%
2035    \ifcase#1%
2036        \or zehntes%
2037        \or zwanzigstes%
2038        \or drei{\ss}igstes%
2039        \or vierzigstes%
2040        \or f\"unfzigstes%
2041        \or sechzigstes%
2042        \or siebzigstes%
2043        \or achtzigstes%
2044        \or neunzigstes%
2045    \fi
2046}%
2047\global\let\@@tenthstringNgerman\@@tenthstringNgerman

    Teens (neuter)
2048\newcommand*\@@teenthstringNgerman[1]{%
2049    \ifcase#1%
2050        zehntes%
2051        \or elftes%
2052        \or zw\"olftes%
2053        \or dreizehntes%
2054        \or vierzehntes%
2055        \or f\"unfzehntes%
2056        \or sechzehntes%

```

```

2057 \or siebzehntes%
2058 \or achtzehntes%
2059 \or neunzehntes%
2060 \fi
2061 }%

```

```

2062 \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.

```

2063 \newcommand*\@@numberunderhundredgerman[2]{%
2064 \ifnum#1<10\relax
2065 \ifnum#1>0\relax
2066 \eappto#2{\@unitstring{#1}}%
2067 \fi
2068 \else
2069 \@tmpstrctr=#1\relax
2070 \@FCmodulo{\@tmpstrctr}{10}%
2071 \ifnum#1<20\relax
2072 \eappto#2{\@teenstring{\@tmpstrctr}}%
2073 \else
2074 \ifnum\@tmpstrctr=0\relax
2075 \else
2076 \eappto#2{\@unitstring{\@tmpstrctr}und}%
2077 \fi
2078 \@tmpstrctr=#1\relax
2079 \divide\@tmpstrctr by 10\relax
2080 \eappto#2{\@tenstring{\@tmpstrctr}}%
2081 \fi
2082 \fi
2083 }%
2084 \global\let\@@numberunderhundredgerman\@@numberunderhundredgerman

```

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

```

2085 \newcommand*\@@numberstringgerman[2]{%
2086 \ifnum#1>99999\relax
2087 \PackageError{fmtcount}{Out of range}%
2088 {This macro only works for values less than 100000}%
2089 \else
2090 \ifnum#1<0\relax
2091 \PackageError{fmtcount}{Negative numbers not permitted}%
2092 {This macro does not work for negative numbers, however
2093 you can try typing "minus" first, and then pass the modulus of
2094 this number}%
2095 \fi
2096 \fi
2097 \def#2{}%
2098 \@strctr=#1\relax \divide\@strctr by 1000\relax
2099 \ifnum\@strctr>1\relax

```

```

#1 is  $\geq 2000$ , \@strctr now contains the number of thousands
2100 \@numberunderhundredgerman{\@strctr}{#2}%
2101 \appto#2{tausend}%
2102 \else
    #1 lies in range [1000,1999]
2103 \ifnum\@strctr=1\relax
2104 \eappto#2{eintausend}%
2105 \fi
2106 \fi
2107 \@strctr=#1\relax
2108 \@FCmodulo{\@strctr}{1000}%
2109 \divide\@strctr by 100\relax
2110 \ifnum\@strctr>1\relax
    now dealing with number in range [200,999]
2111 \eappto#2{\@unitstring{\@strctr}hundert}%
2112 \else
2113 \ifnum\@strctr=1\relax
    dealing with number in range [100,199]
2114 \ifnum#1>1000\relax
    if original number > 1000, use einhundert
2115 \appto#2{einhundert}%
2116 \else
    otherwise use \einhundert
2117 \eappto#2{einhundert}%
2118 \fi
2119 \fi
2120 \fi
2121 \@strctr=#1\relax
2122 \@FCmodulo{\@strctr}{100}%
2123 \ifnum#1=0\relax
2124 \def#2{null}%
2125 \else
2126 \ifnum\@strctr=1\relax
2127 \appto#2{eins}%
2128 \else
2129 \@numberunderhundredgerman{\@strctr}{#2}%
2130 \fi
2131 \fi
2132 }%
2133 \global\let\@numberstringgerman\@numberstringgerman

As above, but for ordinals
2134 \newcommand*\@numberunderhundredthgerman[2]{%
2135 \ifnum#1<10\relax
2136 \eappto#2{\@unitthstring{#1}}%
2137 \else
2138 \@tmpstrctr=#1\relax

```

```

2139 \@FCmodulo{\@tmpstrctr}{10}%
2140 \ifnum#1<20\relax
2141   \eappto#2{\@teenthstring{\@tmpstrctr}}%
2142 \else
2143   \ifnum\@tmpstrctr=0\relax
2144   \else
2145     \eappto#2{\@unitstring{\@tmpstrctr}und}%
2146   \fi
2147   \@tmpstrctr=#1\relax
2148   \divide\@tmpstrctr by 10\relax
2149   \eappto#2{\@tenthstring{\@tmpstrctr}}%
2150 \fi
2151 \fi
2152 }%
2153 \global\let\@@numberunderhundredthgerman\@@numberunderhundredthgerman

2154 \newcommand*{\@@ordinalstringgerman[2]}{%
2155 \@orgargctr=#1\relax
2156 \ifnum\@orgargctr>99999\relax
2157   \PackageError{fmtcount}{Out of range}%
2158   {This macro only works for values less than 100000}%
2159 \else
2160   \ifnum\@orgargctr<0\relax
2161     \PackageError{fmtcount}{Negative numbers not permitted}%
2162     {This macro does not work for negative numbers, however
2163     you can try typing "minus" first, and then pass the modulus of
2164     this number}%
2165   \fi
2166 \fi
2167 \def#2{}%
2168 \@strctr=\@orgargctr\divide\@strctr by 1000\relax
2169 \ifnum\@strctr>1\relax

  #1 is ≥ 2000, \@strctr now contains the number of thousands
2170 \@@numberunderhundredgerman{\@strctr}{#2}%

  is that it, or is there more?
2171 \@tmpstrctr=\@orgargctr\@FCmodulo{\@tmpstrctr}{1000}%
2172 \ifnum\@tmpstrctr=0\relax
2173   \eappto#2{\@thousandth}%
2174 \else
2175   \appto#2{tausend}%
2176 \fi
2177 \else

  #1 lies in range [1000,1999]
2178 \ifnum\@strctr=1\relax
2179   \ifnum\@orgargctr=1000\relax
2180     \eappto#2{\@thousandth}%
2181   \else
2182     \eappto#2{eintausend}%

```

```

2183     \fi
2184 \fi
2185 \fi
2186 \@strctr=\@orgargctr
2187 \@FCmodulo{\@strctr}{100}%
2188 \divide\@strctr by 100\relax
2189 \ifnum\@strctr>1\relax

    now dealing with number in range [200,999] is that it, or is there more?
2190     \@tmpstrctr=\@orgargctr \@FCmodulo{\@tmpstrctr}{100}%
2191     \ifnum\@tmpstrctr=0\relax
2192         \ifnum\@strctr=1\relax
2193             \eappto#2{\@hundredth}%
2194         \else
2195             \eappto#2{\@unitstring{\@strctr}\@hundredth}%
2196         \fi
2197     \else
2198         \eappto#2{\@unitstring{\@strctr}hundert}%
2199     \fi
2200 \else
2201     \ifnum\@strctr=1\relax

        dealing with number in range [100,199] is that it, or is there more?
2202         \@tmpstrctr=\@orgargctr \@FCmodulo{\@tmpstrctr}{100}%
2203         \ifnum\@tmpstrctr=0\relax
2204             \eappto#2{\@hundredth}%
2205         \else
2206             \ifnum\@orgargctr>1000\relax
2207                 \appto#2{einhundert}%
2208             \else
2209                 \eappto#2{einhundert}%
2210             \fi
2211         \fi
2212     \fi
2213 \fi
2214 \@strctr=\@orgargctr
2215 \@FCmodulo{\@strctr}{100}%
2216 \ifthenelse{\@strctr=0 \and \@orgargctr>0 }{}{%
2217 \@@numberunderhundredthgerman{\@strctr}{#2}%
2218 }%
2219 }%
2220 \global\let\@@ordinalstringgerman\@@ordinalstringgerman

    Load fc-germanb.def if not already loaded
2221 \FCloadlang{germanb}%

```

10.1.9 fc-germanb.def

```

2222 \ProvidesFCLanguage{germanb}[2013/08/17]%

    Load fc-german.def if not already loaded

```


2223 \FCloadlang{german}%

Set germanb to be equivalent to german.

2224 \global\let\@ordinalMgermanb=\@ordinalMgerman

2225 \global\let\@ordinalFgermanb=\@ordinalFgerman

2226 \global\let\@ordinalNgermanb=\@ordinalNgerman

2227 \global\let\@numberstringMgermanb=\@numberstringMgerman

2228 \global\let\@numberstringFgermanb=\@numberstringFgerman

2229 \global\let\@numberstringNgermanb=\@numberstringNgerman

2230 \global\let\@NumberstringMgermanb=\@NumberstringMgerman

2231 \global\let\@NumberstringFgermanb=\@NumberstringFgerman

2232 \global\let\@NumberstringNgermanb=\@NumberstringNgerman

2233 \global\let\@ordinalstringMgermanb=\@ordinalstringMgerman

2234 \global\let\@ordinalstringFgermanb=\@ordinalstringFgerman

2235 \global\let\@ordinalstringNgermanb=\@ordinalstringNgerman

2236 \global\let\@OrdinalstringMgermanb=\@OrdinalstringMgerman

2237 \global\let\@OrdinalstringFgermanb=\@OrdinalstringFgerman

2238 \global\let\@OrdinalstringNgermanb=\@OrdinalstringNgerman

10.1.10 fc-italian

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

2239 \ProvidesFCLanguage{italian}[2013/08/17]

2240

2241 \RequirePackage{itnumpar}

2242

2243 \newcommand{\@numberstringMitalian}[2]{%

2244 \edef#2{\noexpand\printnumeroinparole{#1}}%

2245 }

2246 \global\let\@numberstringMitalian\@numberstringMitalian

2247

2248 \newcommand{\@numberstringFitalian}[2]{%

2249 \edef#2{\noexpand\printnumeroinparole{#1}}}%

2250

2251 \global\let\@numberstringFitalian\@numberstringFitalian

2252

2253 \newcommand{\@NumberstringMitalian}[2]{%

2254 \edef#2{\noexpand\printNumeroinparole{#1}}%

2255 }

2256 \global\let\@NumberstringMitalian\@NumberstringMitalian

2257

2258 \newcommand{\@NumberstringFitalian}[2]{%

2259 \edef#2{\noexpand\printNumeroinparole{#1}}%

2260 }

2261 \global\let\@NumberstringFitalian\@NumberstringFitalian

2262

2263 \newcommand{\@ordinalstringMitalian}[2]{%

2264 \edef#2{\noexpand\printordinalem{#1}}%

2265 }

2266 \global\let\@ordinalstringMitalian\@ordinalstringMitalian

2267

```

2268 \newcommand{\@ordinalstringFitalian}[2]{%
2269   \edef#2{\noexpand\printordinalef{#1}}%
2270 }
2271 \global\let\@ordinalstringFitalian\@ordinalstringFitalian
2272
2273 \newcommand{\@ordinalstringMitalian}[2]{%
2274   \edef#2{\noexpand\printOrdinalem{#1}}%
2275 }
2276 \global\let\@ordinalstringMitalian\@ordinalstringMitalian
2277
2278 \newcommand{\@ordinalstringFitalian}[2]{%
2279   \edef#2{\noexpand\printOrdinalef{#1}}%
2280 }
2281 \global\let\@ordinalstringFitalian\@ordinalstringFitalian
2282
2283 \newcommand{\@ordinalMitalian}[2]{%
2284   \edef#2{#1\relax\noexpand\fmtord{o}}%
2285 }
2286 \global\let\@ordinalMitalian\@ordinalMitalian
2287
2288 \newcommand{\@ordinalFitalian}[2]{%
2289   \edef#2{#1\relax\noexpand\fmtord{a}}%
2290 }
2291 \global\let\@ordinalFitalian\@ordinalFitalian

```

10.1.11 fc-ngerman.def

```

2291 \ProvidesFCLanguage{ngerman}[2012/06/18]%
2292 \FCloadlang{german}%
2293 \FCloadlang{ngermanb}%

```

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

```

2294 \global\let\@ordinalMngerman=\@ordinalMgerman
2295 \global\let\@ordinalFngerman=\@ordinalFgerman
2296 \global\let\@ordinalNngerman=\@ordinalNgerman
2297 \global\let\@numberstringMngerman=\@numberstringMgerman
2298 \global\let\@numberstringFngerman=\@numberstringFgerman
2299 \global\let\@numberstringNngerman=\@numberstringNgerman
2300 \global\let\@NumberstringMngerman=\@NumberstringMgerman
2301 \global\let\@NumberstringFngerman=\@NumberstringFgerman
2302 \global\let\@NumberstringNngerman=\@NumberstringNgerman
2303 \global\let\@ordinalstringMngerman=\@ordinalstringMgerman
2304 \global\let\@ordinalstringFngerman=\@ordinalstringFgerman
2305 \global\let\@ordinalstringNngerman=\@ordinalstringNgerman
2306 \global\let\@OrdinalstringMngerman=\@OrdinalstringMgerman
2307 \global\let\@OrdinalstringFngerman=\@OrdinalstringFgerman
2308 \global\let\@OrdinalstringNngerman=\@OrdinalstringNgerman

```

10.1.12 fc-ngermanb.def

```

2309 \ProvidesFCLanguage{ngermanb}[2013/08/17]%

```

2310 \FCloadlang{german}%

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

```
2311 \global\let\@ordinalMngermanb=\@ordinalMgerman
2312 \global\let\@ordinalFngermanb=\@ordinalFgerman
2313 \global\let\@ordinalNngermanb=\@ordinalNgerman
2314 \global\let\@numberstringMngermanb=\@numberstringMgerman
2315 \global\let\@numberstringFngermanb=\@numberstringFgerman
2316 \global\let\@numberstringNngermanb=\@numberstringNgerman
2317 \global\let\@NumberstringMngermanb=\@NumberstringMgerman
2318 \global\let\@NumberstringFngermanb=\@NumberstringFgerman
2319 \global\let\@NumberstringNngermanb=\@NumberstringNgerman
2320 \global\let\@ordinalstringMngermanb=\@ordinalstringMgerman
2321 \global\let\@ordinalstringFngermanb=\@ordinalstringFgerman
2322 \global\let\@ordinalstringNngermanb=\@ordinalstringNgerman
2323 \global\let\@OrdinalstringMngermanb=\@OrdinalstringMgerman
2324 \global\let\@OrdinalstringFngermanb=\@OrdinalstringFgerman
2325 \global\let\@OrdinalstringNngermanb=\@OrdinalstringNgerman
```

Load fc-ngerman.def if not already loaded

2326 \FCloadlang{ngerman}%

10.1.13 fc-portuges.def

Portuguese definitions

2327 \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:

```
2328 \newcommand*\@ordinalMportuges[2]{%
2329   \ifnum#1=0\relax
2330     \edef#2{\number#1}%
2331   \else
2332     \edef#2{\number#1\relax\noexpand\fmtord{o}}%
2333   \fi
2334 }%
2335 \global\let\@ordinalMportuges\@ordinalMportuges
```

Feminine:

```
2336 \newcommand*\@ordinalFportuges[2]{%
2337   \ifnum#1=0\relax
2338     \edef#2{\number#1}%
2339   \else
2340     \edef#2{\number#1\relax\noexpand\fmtord{a}}%
2341   \fi
2342 }%
2343 \global\let\@ordinalFportuges\@ordinalFportuges
```

Make neuter same as masculine:

2344 \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):

```

2345 \newcommand*{\@@unitstringportuges[1]}{
2346   \ifcase#1\relax
2347     zero%
2348     \or um%
2349     \or dois%
2350     \or tr\^es%
2351     \or quatro%
2352     \or cinco%
2353     \or seis%
2354     \or sete%
2355     \or oito%
2356     \or nove%
2357   \fi
2358 }%
2359 \global\let\@@unitstringportuges\@@unitstringportuges
2360 % \end{macrocode}
2361 % As above, but for feminine:
2362 % \begin{macrocode}
2363 \newcommand*{\@@unitstringFportuges[1]}{
2364   \ifcase#1\relax
2365     zero%
2366     \or uma%
2367     \or duas%
2368     \or tr\^es%
2369     \or quatro%
2370     \or cinco%
2371     \or seis%
2372     \or sete%
2373     \or oito%
2374     \or nove%
2375   \fi
2376 }%
2377 \global\let\@@unitstringFportuges\@@unitstringFportuges

```

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

```

2378 \newcommand*{\@@tenstringportuges[1]}{
2379   \ifcase#1\relax
2380     \or dez%
2381     \or vinte%
2382     \or trinta%
2383     \or quarenta%
2384     \or cinq\"uenta%
2385     \or sessenta%
2386     \or setenta%
2387     \or oitenta%
2388     \or noventa%
2389     \or cem%

```

```

2390 \fi
2391}%
2392\global\let\@@tenstringportuges\@@tenstringportuges

  Teens (argument must be a number from 0 to 9):
2393\newcommand*\@@teenstringportuges[1]{%
2394 \ifcase#1\relax
2395   dez%
2396   \or onze%
2397   \or doze%
2398   \or treze%
2399   \or catorze%
2400   \or quinze%
2401   \or dezasseis%
2402   \or dezassete%
2403   \or dezoito%
2404   \or dezanove%
2405 \fi
2406}%
2407\global\let\@@teenstringportuges\@@teenstringportuges

  Hundreds:
2408\newcommand*\@@hundredstringportuges[1]{%
2409 \ifcase#1\relax
2410   \or cento%
2411   \or duzentos%
2412   \or trezentos%
2413   \or quatrocentos%
2414   \or quinhentos%
2415   \or seiscentos%
2416   \or setecentos%
2417   \or oitocentos%
2418   \or novecentos%
2419 \fi
2420}%
2421\global\let\@@hundredstringportuges\@@hundredstringportuges

  Hundreds (feminine):
2422\newcommand*\@@hundredstringFportuges[1]{%
2423 \ifcase#1\relax
2424   \or cento%
2425   \or duzentas%
2426   \or trezentas%
2427   \or quatrocentas%
2428   \or quinhentas%
2429   \or seiscentas%
2430   \or setecentas%
2431   \or oitocentas%
2432   \or novecentas%
2433 \fi
2434}%

```

2435 \global\let\@@hundredstringFportuges\@@hundredstringFportuges

Units (initial letter in upper case):

2436 \newcommand*\@@Unitstringportuges[1]{%

2437 \ifcase#1\relax

2438 Zero%

2439 \or Um%

2440 \or Dois%

2441 \or Tr\^es%

2442 \or Quatro%

2443 \or Cinco%

2444 \or Seis%

2445 \or Sete%

2446 \or Oito%

2447 \or Nove%

2448 \fi

2449 }%

2450 \global\let\@@Unitstringportuges\@@Unitstringportuges

As above, but feminine:

2451 \newcommand*\@@UnitstringFportuges[1]{%

2452 \ifcase#1\relax

2453 Zera%

2454 \or Uma%

2455 \or Duas%

2456 \or Tr\^es%

2457 \or Quatro%

2458 \or Cinco%

2459 \or Seis%

2460 \or Sete%

2461 \or Oito%

2462 \or Nove%

2463 \fi

2464 }%

2465 \global\let\@@UnitstringFportuges\@@UnitstringFportuges

Tens (with initial letter in upper case):

2466 \newcommand*\@@Tenstringportuges[1]{%

2467 \ifcase#1\relax

2468 \or Dez%

2469 \or Vinte%

2470 \or Trinta%

2471 \or Quarenta%

2472 \or Cinq\"uenta%

2473 \or Sessenta%

2474 \or Setenta%

2475 \or Oitenta%

2476 \or Noventa%

2477 \or Cem%

2478 \fi

2479 }%

2480 \global\let\@@Tenstringportuges\@@Tenstringportuges

Teens (with initial letter in upper case):

2481 \newcommand*\@@Teenstringportuges[1]{%

2482 \ifcase#1\relax

2483 Dez%

2484 \or Onze%

2485 \or Doze%

2486 \or Treze%

2487 \or Catorze%

2488 \or Quinze%

2489 \or Dezasseis%

2490 \or Dezassete%

2491 \or Dezoito%

2492 \or Dezanove%

2493 \fi

2494 }%

2495 \global\let\@@Teenstringportuges\@@Teenstringportuges

Hundreds (with initial letter in upper case):

2496 \newcommand*\@@Hundredstringportuges[1]{%

2497 \ifcase#1\relax

2498 \or Cento%

2499 \or Duzentos%

2500 \or Trezentos%

2501 \or Quatrocentos%

2502 \or Quinhentos%

2503 \or Seiscentos%

2504 \or Setecentos%

2505 \or Oitocentos%

2506 \or Novecentos%

2507 \fi

2508 }%

2509 \global\let\@@Hundredstringportuges\@@Hundredstringportuges

As above, but feminine:

2510 \newcommand*\@@HundredstringFportuges[1]{%

2511 \ifcase#1\relax

2512 \or Cento%

2513 \or Duzentas%

2514 \or Trezentas%

2515 \or Quatrocentas%

2516 \or Quinhentas%

2517 \or Seiscentas%

2518 \or Setecentas%

2519 \or Oitocentas%

2520 \or Novecentas%

2521 \fi

2522 }%

2523 \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.)

```

2524 \newcommand*{\@numberstringMportuges}[2]{%
2525   \let\@unitstring=\@unitstringportuges
2526   \let\@teenstring=\@teenstringportuges
2527   \let\@tenstring=\@tenstringportuges
2528   \let\@hundredstring=\@hundredstringportuges
2529   \def\@hundred{cem}\def\@thousand{mil}%
2530   \def\@andname{e}%
2531   \@numberstringportuges{#1}{#2}%
2532 }%
2533 \global\let\@numberstringMportuges\@numberstringMportuges

```

As above, but feminine form:

```

2534 \newcommand*{\@numberstringFportuges}[2]{%
2535   \let\@unitstring=\@unitstringFportuges
2536   \let\@teenstring=\@teenstringportuges
2537   \let\@tenstring=\@tenstringportuges
2538   \let\@hundredstring=\@hundredstringFportuges
2539   \def\@hundred{cem}\def\@thousand{mil}%
2540   \def\@andname{e}%
2541   \@numberstringportuges{#1}{#2}%
2542 }%
2543 \global\let\@numberstringFportuges\@numberstringFportuges

```

Make neuter same as masculine:

```

2544 \global\let\@numberstringNportuges\@numberstringMportuges

```

As above, but initial letters in upper case:

```

2545 \newcommand*{\@NumberstringMportuges}[2]{%
2546   \let\@unitstring=\@Unitstringportuges
2547   \let\@teenstring=\@Teenstringportuges
2548   \let\@tenstring=\@Tenstringportuges
2549   \let\@hundredstring=\@Hundredstringportuges
2550   \def\@hundred{Cem}\def\@thousand{Mil}%
2551   \def\@andname{e}%
2552   \@numberstringportuges{#1}{#2}%
2553 }%
2554 \global\let\@NumberstringMportuges\@NumberstringMportuges

```

As above, but feminine form:

```

2555 \newcommand*{\@NumberstringFportuges}[2]{%
2556   \let\@unitstring=\@UnitstringFportuges
2557   \let\@teenstring=\@Teenstringportuges
2558   \let\@tenstring=\@Tenstringportuges
2559   \let\@hundredstring=\@HundredstringFportuges
2560   \def\@hundred{Cem}\def\@thousand{Mil}%
2561   \def\@andname{e}%
2562   \@numberstringportuges{#1}{#2}%
2563 }%

```


2564 \global\let\@NumberstringFportuges\@NumberstringFportuges

Make neuter same as masculine:

2565 \global\let\@NumberstringNportuges\@NumberstringMportuges

As above, but for ordinals.

2566 \newcommand*{\@ordinalstringMportuges}[2]{%

2567 \let\@unitthstring=\@unitthstringportuges

2568 \let\@unitstring=\@unitstringportuges

2569 \let\@teenthstring=\@teenthstringportuges

2570 \let\@tenthstring=\@tenthstringportuges

2571 \let\@hundredthstring=\@hundredthstringportuges

2572 \def\@thousandth{mil\’esimo}%

2573 \@ordinalstringportuges{#1}{#2}%

2574 }%

2575 \global\let\@ordinalstringMportuges\@ordinalstringMportuges

Feminine form:

2576 \newcommand*{\@ordinalstringFportuges}[2]{%

2577 \let\@unitthstring=\@unitthstringFportuges

2578 \let\@unitstring=\@unitstringFportuges

2579 \let\@teenthstring=\@teenthstringFportuges

2580 \let\@tenthstring=\@tenthstringFportuges

2581 \let\@hundredthstring=\@hundredthstringFportuges

2582 \def\@thousandth{mil\’esima}%

2583 \@ordinalstringportuges{#1}{#2}%

2584 }%

2585 \global\let\@ordinalstringFportuges\@ordinalstringFportuges

Make neuter same as masculine:

2586 \global\let\@ordinalstringNportuges\@ordinalstringMportuges

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

2587 \newcommand*{\@OrdinalstringMportuges}[2]{%

2588 \let\@unitthstring=\@Unitthstringportuges

2589 \let\@unitstring=\@Unitstringportuges

2590 \let\@teenthstring=\@Teenthstringportuges

2591 \let\@tenthstring=\@Tenthstringportuges

2592 \let\@hundredthstring=\@Hundredthstringportuges

2593 \def\@thousandth{Mil\’esimo}%

2594 \@ordinalstringportuges{#1}{#2}%

2595 }%

2596 \global\let\@OrdinalstringMportuges\@OrdinalstringMportuges

Feminine form:

2597 \newcommand*{\@OrdinalstringFportuges}[2]{%

2598 \let\@unitthstring=\@UnitthstringFportuges

2599 \let\@unitstring=\@UnitstringFportuges

2600 \let\@teenthstring=\@Teenthstringportuges

2601 \let\@tenthstring=\@TenthstringFportuges

2602 \let\@hundredthstring=\@HundredthstringFportuges

2603 \def\@thousandth{Mil\’esima}%

```

2604 \@@ordinalstringportuges{#1}{#2}%
2605 }%
2606 \global\let\@OrdinalstringFportuges\@OrdinalstringFportuges

    Make neuter same as masculine:
2607 \global\let\@OrdinalstringMportuges\@OrdinalstringMportuges

    In order to do the ordinals, split into units, teens, tens and hundreds. Units:
2608 \newcommand*\@@unitthstringportuges[1]{%
2609     \ifcase#1\relax
2610         zero%
2611         \or primeiro%
2612         \or segundo%
2613         \or terceiro%
2614         \or quarto%
2615         \or quinto%
2616         \or sexto%
2617         \or s\'etimo%
2618         \or oitavo%
2619         \or nono%
2620     \fi
2621 }%
2622 \global\let\@@unitthstringportuges\@@unitthstringportuges

    Tens:
2623 \newcommand*\@@tenthstringportuges[1]{%
2624     \ifcase#1\relax
2625         \or d\'ecimo%
2626         \or vig\'esimo%
2627         \or trig\'esimo%
2628         \or quadrag\'esimo%
2629         \or q\'uinquag\'esimo%
2630         \or sexag\'esimo%
2631         \or setuag\'esimo%
2632         \or octog\'esimo%
2633         \or nonag\'esimo%
2634     \fi
2635 }%
2636 \global\let\@@tenthstringportuges\@@tenthstringportuges

    Teens:
2637 \newcommand*\@@teenthstringportuges[1]{%
2638     \@tenthstring{1}%
2639     \ifnum#1>0\relax
2640         -\@unitthstring{#1}%
2641     \fi
2642 }%
2643 \global\let\@@teenthstringportuges\@@teenthstringportuges

    Hundreds:
2644 \newcommand*\@@hundredthstringportuges[1]{%

```

```

2645 \ifcase#1\relax
2646 \or cent\'esimo%
2647 \or ducent\'esimo%
2648 \or trecent\'esimo%
2649 \or quadringent\'esimo%
2650 \or q\'uingent\'esimo%
2651 \or seiscent\'esimo%
2652 \or setingent\'esimo%
2653 \or octingent\'esimo%
2654 \or nongent\'esimo%
2655 \fi
2656 }%
2657 \global\let\@@hundredthstringportuges\@@hundredthstringportuges

```

Units (feminine):

```

2658 \newcommand*\@@unitthstringFportuges[1]{%
2659 \ifcase#1\relax
2660 zero%
2661 \or primeira%
2662 \or segunda%
2663 \or terceira%
2664 \or quarta%
2665 \or quinta%
2666 \or sexta%
2667 \or s\'etima%
2668 \or oitava%
2669 \or nona%
2670 \fi
2671 }%
2672 \global\let\@@unitthstringFportuges\@@unitthstringFportuges

```

Tens (feminine):

```

2673 \newcommand*\@@tenthstringFportuges[1]{%
2674 \ifcase#1\relax
2675 \or d\'ecima%
2676 \or vig\'esima%
2677 \or trig\'esima%
2678 \or quadrag\'esima%
2679 \or q\'uinquag\'esima%
2680 \or sexag\'esima%
2681 \or setuag\'esima%
2682 \or octog\'esima%
2683 \or nonag\'esima%
2684 \fi
2685 }%
2686 \global\let\@@tenthstringFportuges\@@tenthstringFportuges

```

Hundreds (feminine):

```

2687 \newcommand*\@@hundredthstringFportuges[1]{%
2688 \ifcase#1\relax
2689 \or cent\'esima%

```

```

2690 \or ducent\'esima%
2691 \or trecent\'esima%
2692 \or quadringent\'esima%
2693 \or q\'uigent\'esima%
2694 \or seiscent\'esima%
2695 \or setingent\'esima%
2696 \or octingent\'esima%
2697 \or nongent\'esima%
2698 \fi
2699 }%
2700 \global\let\@@hundredthstringFportuges\@@hundredthstringFportuges

```

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

```

2701 \newcommand*\@@Unitthstringportuges[1]{%
2702 \ifcase#1\relax
2703 Zero%
2704 \or Primeiro%
2705 \or Segundo%
2706 \or Terceiro%
2707 \or Quarto%
2708 \or Quinto%
2709 \or Sexto%
2710 \or S\'etimo%
2711 \or Oitavo%
2712 \or Nono%
2713 \fi
2714 }%
2715 \global\let\@@Unitthstringportuges\@@Unitthstringportuges

```

Tens:

```

2716 \newcommand*\@@Tenthstringportuges[1]{%
2717 \ifcase#1\relax
2718 \or D\'ecimo%
2719 \or Vig\'esimo%
2720 \or Trig\'esimo%
2721 \or Quadrag\'esimo%
2722 \or Q\'uinquag\'esimo%
2723 \or Sexag\'esimo%
2724 \or Setuag\'esimo%
2725 \or Octog\'esimo%
2726 \or Nonag\'esimo%
2727 \fi
2728 }%
2729 \global\let\@@Tenthstringportuges\@@Tenthstringportuges

```

Hundreds:

```

2730 \newcommand*\@@Hundredthstringportuges[1]{%
2731 \ifcase#1\relax
2732 \or Cent\'esimo%
2733 \or Ducent\'esimo%
2734 \or Trecent\'esimo%

```

```

2735 \or Quadringent\'esimo%
2736 \or Q\"uingent\'esimo%
2737 \or Seiscent\'esimo%
2738 \or Setingent\'esimo%
2739 \or Octingent\'esimo%
2740 \or Nongent\'esimo%
2741 \fi
2742 }%
2743 \global\let\@@Hundredthstringportuges\@@Hundredthstringportuges

```

As above, but feminine. Units:

```

2744 \newcommand*\@@UnitthstringFportuges[1]{%
2745 \ifcase#1\relax
2746 Zera%
2747 \or Primeira%
2748 \or Segunda%
2749 \or Terceira%
2750 \or Quarta%
2751 \or Quinta%
2752 \or Sexta%
2753 \or S\'etima%
2754 \or Oitava%
2755 \or Nona%
2756 \fi
2757 }%
2758 \global\let\@@UnitthstringFportuges\@@UnitthstringFportuges

```

Tens (feminine);

```

2759 \newcommand*\@@TenthstringFportuges[1]{%
2760 \ifcase#1\relax
2761 \or D\'ecima%
2762 \or Vig\'esima%
2763 \or Trig\'esima%
2764 \or Quadrag\'esima%
2765 \or Q\"uinquag\'esima%
2766 \or Sexag\'esima%
2767 \or Setuag\'esima%
2768 \or Octog\'esima%
2769 \or Nonag\'esima%
2770 \fi
2771 }%
2772 \global\let\@@TenthstringFportuges\@@TenthstringFportuges

```

Hundreds (feminine):

```

2773 \newcommand*\@@HundredthstringFportuges[1]{%
2774 \ifcase#1\relax
2775 \or Cent\'esima%
2776 \or Ducent\'esima%
2777 \or Trecent\'esima%
2778 \or Quadringent\'esima%
2779 \or Q\"uingent\'esima%

```

```

2780 \or Seiscent\'esima%
2781 \or Setingent\'esima%
2782 \or Octingent\'esima%
2783 \or Nongent\'esima%
2784 \fi
2785 }%
2786 \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.)

```

2787 \newcommand*{\@@numberstringportuges[2]}{%
2788 \ifnum#1>99999\relax
2789 \PackageError{fmtcount}{Out of range}%
2790 {This macro only works for values less than 100000}%
2791 \else
2792 \ifnum#1<0\relax
2793 \PackageError{fmtcount}{Negative numbers not permitted}%
2794 {This macro does not work for negative numbers, however
2795 you can try typing "minus" first, and then pass the modulus of
2796 this number}%
2797 \fi
2798 \fi
2799 \def#2{%
2800 \@strctr=#1\relax \divide\@strctr by 1000\relax
2801 \ifnum\@strctr>9\relax
2802 \divide\@strctr by 10\relax
2803 \ifnum\@strctr>1\relax
2804 \let\@@fc@numstr#2\relax
2805 \protected@edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
2806 \@strctr=#1 \divide\@strctr by 1000\relax
2807 \@FCmodulo{\@strctr}{10}%
2808 \ifnum\@strctr>0
2809 \ifnum\@strctr=1\relax
2810 \let\@@fc@numstr#2\relax
2811 \protected@edef#2{\@@fc@numstr\ \@andname}%
2812 \fi
2813 \let\@@fc@numstr#2\relax
2814 \protected@edef#2{\@@fc@numstr\ \@unitstring{\@strctr}}%
2815 \fi
2816 \else
2817 \@strctr=#1\relax
2818 \divide\@strctr by 1000\relax
2819 \@FCmodulo{\@strctr}{10}%
2820 \let\@@fc@numstr#2\relax
2821 \protected@edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
2822 \fi

```

```

2823 \let\@@fc@numstr#2\relax
2824 \protected@edef#2{\@@fc@numstr\ \@thousand}%
2825 \else
2826 \ifnum\@strctr>0\relax
2827 \ifnum\@strctr>1\relax
2828 \let\@@fc@numstr#2\relax
2829 \protected@edef#2{\@@fc@numstr\@unitstring{\@strctr}\ }%
2830 \fi
2831 \let\@@fc@numstr#2\relax
2832 \protected@edef#2{\@@fc@numstr\@thousand}%
2833 \fi
2834 \fi
2835 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
2836 \divide\@strctr by 100\relax
2837 \ifnum\@strctr>0\relax
2838 \ifnum#1>1000 \relax
2839 \let\@@fc@numstr#2\relax
2840 \protected@edef#2{\@@fc@numstr\ }%
2841 \fi
2842 \@tmpstrctr=#1\relax
2843 \@FCmodulo{\@tmpstrctr}{1000}%
2844 \let\@@fc@numstr#2\relax
2845 \ifnum\@tmpstrctr=100\relax
2846 \protected@edef#2{\@@fc@numstr\@tenstring{10}}%
2847 \else
2848 \protected@edef#2{\@@fc@numstr\@hundredstring{\@strctr}}%
2849 \fi%
2850 \fi
2851 \@strctr=#1\relax \@FCmodulo{\@strctr}{100}%
2852 \ifnum#1>100\relax
2853 \ifnum\@strctr>0\relax
2854 \let\@@fc@numstr#2\relax
2855 \protected@edef#2{\@@fc@numstr\ \@andname\ }%
2856 \fi
2857 \fi
2858 \ifnum\@strctr>19\relax
2859 \divide\@strctr by 10\relax
2860 \let\@@fc@numstr#2\relax
2861 \protected@edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
2862 \@strctr=#1\relax \@FCmodulo{\@strctr}{10}%
2863 \ifnum\@strctr>0
2864 \ifnum\@strctr=1\relax
2865 \let\@@fc@numstr#2\relax
2866 \protected@edef#2{\@@fc@numstr\ \@andname}%
2867 \else
2868 \ifnum#1>100\relax
2869 \let\@@fc@numstr#2\relax
2870 \protected@edef#2{\@@fc@numstr\ \@andname}%
2871 \fi

```

```

2872 \fi
2873 \let\@@fc@numstr#2\relax
2874 \protected@edef#2{\@@fc@numstr\ \unitstring{\@strctr}}%
2875 \fi
2876 \else
2877 \ifnum\@strctr<10\relax
2878 \ifnum\@strctr=0\relax
2879 \ifnum#1<100\relax
2880 \let\@@fc@numstr#2\relax
2881 \protected@edef#2{\@@fc@numstr\unitstring{\@strctr}}%
2882 \fi
2883 \else %(>0,<10)
2884 \let\@@fc@numstr#2\relax
2885 \protected@edef#2{\@@fc@numstr\unitstring{\@strctr}}%
2886 \fi
2887 \else%>10
2888 \@FCmodulo{\@strctr}{10}%
2889 \let\@@fc@numstr#2\relax
2890 \protected@edef#2{\@@fc@numstr\teenstring{\@strctr}}%
2891 \fi
2892 \fi
2893 }%
2894 \global\let\@@numberstringportuges\@@numberstringportuges

```

As above, but for ordinals.

```

2895 \newcommand*\@@ordinalstringportuges[2]{%
2896 \@strctr=#1\relax
2897 \ifnum#1>99999
2898 \PackageError{fmtcount}{Out of range}%
2899 {This macro only works for values less than 100000}%
2900 \else
2901 \ifnum#1<0
2902 \PackageError{fmtcount}{Negative numbers not permitted}%
2903 {This macro does not work for negative numbers, however
2904 you can try typing "minus" first, and then pass the modulus of
2905 this number}%
2906 \else
2907 \def#2{}%
2908 \ifnum\@strctr>999\relax
2909 \divide\@strctr by 1000\relax
2910 \ifnum\@strctr>1\relax
2911 \ifnum\@strctr>9\relax
2912 \@tmpstrctr=\@strctr
2913 \ifnum\@strctr<20
2914 \@FCmodulo{\@tmpstrctr}{10}%
2915 \let\@@fc@ordstr#2\relax
2916 \protected@edef#2{\@@fc@ordstr\teenthsstring{\@tmpstrctr}}%
2917 \else
2918 \divide\@tmpstrctr by 10\relax
2919 \let\@@fc@ordstr#2\relax

```



```

2920         \protected@edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
2921         \@tmpstrctr=\@strctr
2922         \@FCmodulo{\@tmpstrctr}{10}%
2923         \ifnum\@tmpstrctr>0\relax
2924             \let\@@fc@ordstr#2\relax
2925             \protected@edef#2{\@@fc@ordstr\@unitthstring{\@tmpstrctr}}%
2926         \fi
2927     \fi
2928 \else
2929     \let\@@fc@ordstr#2\relax
2930     \protected@edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
2931 \fi
2932 \fi
2933 \let\@@fc@ordstr#2\relax
2934 \protected@edef#2{\@@fc@ordstr\@thousandth}%
2935 \fi
2936 \@strctr=#1\relax
2937 \@FCmodulo{\@strctr}{1000}%
2938 \ifnum\@strctr>99\relax
2939     \@tmpstrctr=\@strctr
2940     \divide\@tmpstrctr by 100\relax
2941     \ifnum#1>1000\relax
2942         \let\@@fc@ordstr#2\relax
2943         \protected@edef#2{\@@fc@ordstr-}%
2944     \fi
2945     \let\@@fc@ordstr#2\relax
2946     \protected@edef#2{\@@fc@ordstr\@hundredthstring{\@tmpstrctr}}%
2947 \fi
2948 \@FCmodulo{\@strctr}{100}%
2949 \ifnum#1>99\relax
2950     \ifnum\@strctr>0\relax
2951         \let\@@fc@ordstr#2\relax
2952         \protected@edef#2{\@@fc@ordstr-}%
2953     \fi
2954 \fi
2955 \ifnum\@strctr>9\relax
2956     \@tmpstrctr=\@strctr
2957     \divide\@tmpstrctr by 10\relax
2958     \let\@@fc@ordstr#2\relax
2959     \protected@edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
2960     \@tmpstrctr=\@strctr
2961     \@FCmodulo{\@tmpstrctr}{10}%
2962     \ifnum\@tmpstrctr>0\relax
2963         \let\@@fc@ordstr#2\relax
2964         \protected@edef#2{\@@fc@ordstr-\@unitthstring{\@tmpstrctr}}%
2965     \fi
2966 \else
2967     \ifnum\@strctr=0\relax
2968         \ifnum#1=0\relax

```

```

2969      \let\@fc@ordstr#2\relax
2970      \protected@edef#2{\@fc@ordstr\@unitstring{0}}%
2971      \fi
2972    \else
2973      \let\@fc@ordstr#2\relax
2974      \protected@edef#2{\@fc@ordstr\@unitthstring{\@strctr}}%
2975      \fi
2976    \fi
2977  \fi
2978  \fi
2979 }%
2980 \global\let\@ordinalstringportuges\@ordinalstringportuges

```

10.1.14 fc-portuguese.def

```

2981 \ProvidesFCLanguage{portuguese}[2014/06/09]%

```

Load fc-portuges.def if not already loaded.

```

2982 \FCloadlang{portuges}%

```

Set portuguese to be equivalent to portuges.

```

2983 \global\let\@ordinalMportuguese=\@ordinalMportuges
2984 \global\let\@ordinalFportuguese=\@ordinalFportuges
2985 \global\let\@ordinalNportuguese=\@ordinalNportuges
2986 \global\let\@numberstringMportuguese=\@numberstringMportuges
2987 \global\let\@numberstringFportuguese=\@numberstringFportuges
2988 \global\let\@numberstringNportuguese=\@numberstringNportuges
2989 \global\let\@NumberstringMportuguese=\@NumberstringMportuges
2990 \global\let\@NumberstringFportuguese=\@NumberstringFportuges
2991 \global\let\@NumberstringNportuguese=\@NumberstringNportuges
2992 \global\let\@ordinalstringMportuguese=\@ordinalstringMportuges
2993 \global\let\@ordinalstringFportuguese=\@ordinalstringFportuges
2994 \global\let\@ordinalstringNportuguese=\@ordinalstringNportuges
2995 \global\let\@OrdinalstringMportuguese=\@OrdinalstringMportuges
2996 \global\let\@OrdinalstringFportuguese=\@OrdinalstringFportuges
2997 \global\let\@OrdinalstringNportuguese=\@OrdinalstringNportuges

```

10.1.15 fc-spanish.def

Spanish definitions

```

2998 \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:

```

2999 \newcommand*\@ordinalMspanish[2]{%
3000   \edef#2{\number#1\relax\noexpand\fmtord{o}}%
3001 }%
3002 \global\let\@ordinalMspanish\@ordinalMspanish

```

Feminine:

```

3003 \newcommand*\@ordinalFspanish[2]{%
3004   \edef#2{\number#1\relax\noexpand\fmtord{a}}%

```

```

3005}%
3006\global\let\@ordinalFspanish\@ordinalFspanish

  Make neuter same as masculine:
3007\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):
3008\newcommand*\@@unitstringspanish[1]{%
3009  \ifcase#1\relax
3010    cero%
3011    \or uno%
3012    \or dos%
3013    \or tres%
3014    \or cuatro%
3015    \or cinco%
3016    \or seis%
3017    \or siete%
3018    \or ocho%
3019    \or nueve%
3020  \fi
3021}%
3022\global\let\@@unitstringspanish\@@unitstringspanish

  Feminine:
3023\newcommand*\@@unitstringFspanish[1]{%
3024  \ifcase#1\relax
3025    cera%
3026    \or una%
3027    \or dos%
3028    \or tres%
3029    \or cuatro%
3030    \or cinco%
3031    \or seis%
3032    \or siete%
3033    \or ocho%
3034    \or nueve%
3035  \fi
3036}%
3037\global\let\@@unitstringFspanish\@@unitstringFspanish

  Tens (argument must go from 1 to 10):
3038\newcommand*\@@tenstringspanish[1]{%
3039  \ifcase#1\relax
3040    \or diez%
3041    \or veinte%
3042    \or treinta%
3043    \or cuarenta%
3044    \or cincuenta%
3045    \or sesenta%
3046    \or setenta%

```

```

3047     \or ochenta%
3048     \or noventa%
3049     \or cien%
3050 \fi
3051}%
3052\global\let\@@tenstringspanish\@@tenstringspanish

    Teens:
3053\newcommand*\@@teenstringspanish[1]{%
3054 \ifcase#1\relax
3055     diez%
3056     \or once%
3057     \or doce%
3058     \or trece%
3059     \or catorce%
3060     \or quince%
3061     \or diecis\'eis%
3062     \or diecisiete%
3063     \or dieciocho%
3064     \or diecinueve%
3065 \fi
3066}%
3067\global\let\@@teenstringspanish\@@teenstringspanish

    Twenties:
3068\newcommand*\@@twentystringspanish[1]{%
3069 \ifcase#1\relax
3070     veinte%
3071     \or veintiuno%
3072     \or veintid\'os%
3073     \or veintitr\'es%
3074     \or veinticuatro%
3075     \or veinticinco%
3076     \or veintis\'eis%
3077     \or veintisiete%
3078     \or veintiocho%
3079     \or veintinueve%
3080 \fi
3081}%
3082\global\let\@@twentystringspanish\@@twentystringspanish

    Feminine form:
3083\newcommand*\@@twentystringFspanish[1]{%
3084 \ifcase#1\relax
3085     veinte%
3086     \or veintiuna%
3087     \or veintid\'os%
3088     \or veintitr\'es%
3089     \or veinticuatro%
3090     \or veinticinco%
3091     \or veintis\'eis%

```

```

3092     \or veintisiete%
3093     \or veintiocho%
3094     \or veintinueve%
3095 \fi
3096}%
3097\global\let\@@twentystringFspanish\@@twentystringFspanish

```

Hundreds:

```

3098\newcommand*\@@hundredstringspanish[1]{%
3099 \ifcase#1\relax
3100     \or ciento%
3101     \or doscientos%
3102     \or trescientos%
3103     \or cuatrocientos%
3104     \or quinientos%
3105     \or seiscientos%
3106     \or setecientos%
3107     \or ochocientos%
3108     \or novecientos%
3109 \fi
3110}%
3111\global\let\@@hundredstringspanish\@@hundredstringspanish

```

Feminine form:

```

3112\newcommand*\@@hundredstringFspanish[1]{%
3113 \ifcase#1\relax
3114     \or cienta%
3115     \or doscientas%
3116     \or trescientas%
3117     \or cuatrocientas%
3118     \or quinientas%
3119     \or seiscientas%
3120     \or setecientas%
3121     \or ochocientas%
3122     \or novecientas%
3123 \fi
3124}%
3125\global\let\@@hundredstringFspanish\@@hundredstringFspanish

```

As above, but with initial letter uppercase:

```

3126\newcommand*\@@Unitstringspanish[1]{%
3127 \ifcase#1\relax
3128     Cero%
3129     \or Uno%
3130     \or Dos%
3131     \or Tres%
3132     \or Cuatro%
3133     \or Cinco%
3134     \or Seis%
3135     \or Siete%
3136     \or Ocho%

```

```

3137     \or Nueve%
3138   \fi
3139 }%
3140 \global\let\@@Unitstringspanish\@@Unitstringspanish

```

Feminine form:

```

3141 \newcommand*\@@UnitstringFspanish[1]{%
3142   \ifcase#1\relax
3143     Cera%
3144     \or Una%
3145     \or Dos%
3146     \or Tres%
3147     \or Cuatro%
3148     \or Cinco%
3149     \or Seis%
3150     \or Siete%
3151     \or Ocho%
3152     \or Nueve%
3153   \fi
3154 }%
3155 \global\let\@@UnitstringFspanish\@@UnitstringFspanish

```

Tens:

```

3156 %\changes{2.0}{2012-06-18}{fixed spelling mistake (correction
3157 %provided by Fernando Maldonado)}
3158 \newcommand*\@@Tenstringspanish[1]{%
3159   \ifcase#1\relax
3160     \or Diez%
3161     \or Veinte%
3162     \or Treinta%
3163     \or Cuarenta%
3164     \or Cincuenta%
3165     \or Sesenta%
3166     \or Setenta%
3167     \or Ochenta%
3168     \or Noventa%
3169     \or Cien%
3170   \fi
3171 }%
3172 \global\let\@@Tenstringspanish\@@Tenstringspanish

```

Teens:

```

3173 \newcommand*\@@Teenstringspanish[1]{%
3174   \ifcase#1\relax
3175     Diez%
3176     \or Once%
3177     \or Doce%
3178     \or Trece%
3179     \or Catorce%
3180     \or Quince%
3181     \or Diecis\'eis%

```

```

3182     \or Diecisiete%
3183     \or Dieciocho%
3184     \or Diecinueve%
3185     \fi
3186 }%
3187 \global\let\@@Teenstringspanish\@@Teenstringspanish

```

Twenties:

```

3188 \newcommand*\@@Twentystringspanish[1]{%
3189   \ifcase#1\relax
3190     Veinte%
3191     \or Veintiuno%
3192     \or Veintid\'os%
3193     \or Veintitr\'es%
3194     \or Veinticuatro%
3195     \or Veinticinco%
3196     \or Veintis\'eis%
3197     \or Veintisiete%
3198     \or Veintiocho%
3199     \or Veintinueve%
3200   \fi
3201 }%
3202 \global\let\@@Twentystringspanish\@@Twentystringspanish

```

Feminine form:

```

3203 \newcommand*\@@TwentystringFspanish[1]{%
3204   \ifcase#1\relax
3205     Veinte%
3206     \or Veintiuna%
3207     \or Veintid\'os%
3208     \or Veintitr\'es%
3209     \or Veinticuatro%
3210     \or Veinticinco%
3211     \or Veintis\'eis%
3212     \or Veintisiete%
3213     \or Veintiocho%
3214     \or Veintinueve%
3215   \fi
3216 }%
3217 \global\let\@@TwentystringFspanish\@@TwentystringFspanish

```

Hundreds:

```

3218 \newcommand*\@@Hundredstringspanish[1]{%
3219   \ifcase#1\relax
3220     \or Ciento%
3221     \or Doscientos%
3222     \or Trescientos%
3223     \or Cuatrocientos%
3224     \or Quinientos%
3225     \or Seiscientos%
3226     \or Setecientos%

```

```

3227 \or Ochocientos%
3228 \or Novecientos%
3229 \fi
3230 }%
3231 \global\let\@@Hundredstringspanish\@@Hundredstringspanish

```

Feminine form:

```

3232 \newcommand*\@@HundredstringFspanish[1]{%
3233 \ifcase#1\relax
3234 \or Cienta%
3235 \or Doscientas%
3236 \or Trescientas%
3237 \or Cuatrocientas%
3238 \or Quinientas%
3239 \or Seiscientas%
3240 \or Setecientas%
3241 \or Ochocientas%
3242 \or Novecientas%
3243 \fi
3244 }%
3245 \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.)

```

3246 \newcommand*\@numberstringMspanish[2]{%
3247 \let\@unitstring=\@unitstringspanish
3248 \let\@teenstring=\@teenstringspanish
3249 \let\@tenstring=\@tenstringspanish
3250 \let\@twentystring=\@twentystringspanish
3251 \let\@hundredstring=\@hundredstringspanish
3252 \def\@hundred{cien}\def\@thousand{mil}%
3253 \def\@andname{y}%
3254 \@numberstringspanish{#1}{#2}%
3255 }%
3256 \global\let\@numberstringMspanish\@numberstringMspanish

```

Feminine form:

```

3257 \newcommand*\@numberstringFspanish[2]{%
3258 \let\@unitstring=\@unitstringFspanish
3259 \let\@teenstring=\@teenstringspanish
3260 \let\@tenstring=\@tenstringspanish
3261 \let\@twentystring=\@twentystringFspanish
3262 \let\@hundredstring=\@hundredstringFspanish
3263 \def\@hundred{cien}\def\@thousand{mil}%
3264 \def\@andname{b}%
3265 \@numberstringspanish{#1}{#2}%
3266 }%
3267 \global\let\@numberstringFspanish\@numberstringFspanish

```

Make neuter same as masculine:

3268 \global\let\@NumberstringNspanish\@NumberstringMspanish

As above, but initial letters in upper case:

```
3269 \newcommand*\@NumberstringMspanish}[2]{%
3270   \let\@unitstring=\@Unitstringspanish
3271   \let\@teenstring=\@Teenstringspanish
3272   \let\@tenstring=\@Tenstringspanish
3273   \let\@twentystring=\@Twentystringspanish
3274   \let\@hundredstring=\@Hundredstringspanish
3275   \def\@andname{y}%
3276   \def\@hundred{Cien}\def\@thousand{Mil}%
3277   \@numberstringspanish{#1}{#2}%
3278 }%
3279 \global\let\@NumberstringMspanish\@NumberstringMspanish
```

Feminine form:

```
3280 \newcommand*\@NumberstringFspanish}[2]{%
3281   \let\@unitstring=\@UnitstringFspanish
3282   \let\@teenstring=\@Teenstringspanish
3283   \let\@tenstring=\@Tenstringspanish
3284   \let\@twentystring=\@TwentystringFspanish
3285   \let\@hundredstring=\@HundredstringFspanish
3286   \def\@andname{b}%
3287   \def\@hundred{Cien}\def\@thousand{Mil}%
3288   \@numberstringspanish{#1}{#2}%
3289 }%
3290 \global\let\@NumberstringFspanish\@NumberstringFspanish
```

Make neuter same as masculine:

3291 \global\let\@NumberstringNspanish\@NumberstringMspanish

As above, but for ordinals.

```
3292 \newcommand*\@OrdinalstringMspanish}[2]{%
3293   \let\@unitthstring=\@unitthstringspanish
3294   \let\@unitstring=\@unitstringspanish
3295   \let\@teenthstring=\@teenthstringspanish
3296   \let\@tenthstring=\@tenthstringspanish
3297   \let\@hundredthstring=\@hundredthstringspanish
3298   \def\@thousandth{mil'esimo}%
3299   \@ordinalstringspanish{#1}{#2}%
3300 }%
3301 \global\let\@OrdinalstringMspanish\@OrdinalstringMspanish
```

Feminine form:

```
3302 \newcommand*\@OrdinalstringFspanish}[2]{%
3303   \let\@unitthstring=\@unitthstringFspanish
3304   \let\@unitstring=\@unitstringFspanish
3305   \let\@teenthstring=\@teenthstringFspanish
3306   \let\@tenthstring=\@tenthstringFspanish
3307   \let\@hundredthstring=\@hundredthstringFspanish
3308   \def\@thousandth{mil'esima}%

```

```

3309 \@@ordinalstringspanish{#1}{#2}%
3310 }%
3311 \global\let\@ordinalstringFspanish\@ordinalstringFspanish
    Make neuter same as masculine:
3312 \global\let\@ordinalstringMspanish\@ordinalstringMspanish
    As above, but with initial letters in upper case.
3313 \newcommand*{\@OrdinalstringMspanish}[2]{%
3314 \let\@unitthstring=\@Unitthstringspanish
3315 \let\@unitstring=\@Unitstringspanish
3316 \let\@teenthstring=\@Teenthstringspanish
3317 \let\@tenthstring=\@Tenthstringspanish
3318 \let\@hundredthstring=\@Hundredthstringspanish
3319 \def\@thousandth{Mil\'esimo}%
3320 \@@ordinalstringspanish{#1}{#2}%
3321 }
3322 \global\let\@OrdinalstringMspanish\@OrdinalstringMspanish
    Feminine form:
3323 \newcommand*{\@OrdinalstringFspanish}[2]{%
3324 \let\@unitthstring=\@UnitthstringFspanish
3325 \let\@unitstring=\@UnitstringFspanish
3326 \let\@teenthstring=\@TeenthstringFspanish
3327 \let\@tenthstring=\@TenthstringFspanish
3328 \let\@hundredthstring=\@HundredthstringFspanish
3329 \def\@thousandth{Mil\'esima}%
3330 \@@ordinalstringspanish{#1}{#2}%
3331 }%
3332 \global\let\@OrdinalstringFspanish\@OrdinalstringFspanish
    Make neuter same as masculine:
3333 \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:
3334 \newcommand*\@unitthstringspanish[1]{%
3335 \ifcase#1\relax
3336     cero%
3337     \or primero%
3338     \or segundo%
3339     \or tercero%
3340     \or cuarto%
3341     \or quinto%
3342     \or sexto%
3343     \or s\'eptimo%
3344     \or octavo%
3345     \or noveno%
3346 \fi
3347 }%
3348 \global\let\@unitthstringspanish\@unitthstringspanish

```

Tens:

```
3349 \newcommand*{\@@tenthstringspanish[1]}{%
3350   \ifcase#1\relax
3351     \or d\'ecimo%
3352     \or vig\'esimo%
3353     \or trig\'esimo%
3354     \or cuadrag\'esimo%
3355     \or quincuag\'esimo%
3356     \or sexag\'esimo%
3357     \or septuag\'esimo%
3358     \or octog\'esimo%
3359     \or nonag\'esimo%
3360   \fi
3361}%
3362 \global\let\@@tenthstringspanish\@@tenthstringspanish
```

Teens:

```
3363 \newcommand*{\@@teenthstringspanish[1]}{%
3364   \ifcase#1\relax
3365     d\'ecimo%
3366     \or und\'ecimo%
3367     \or duod\'ecimo%
3368     \or decimotercero%
3369     \or decimocuarto%
3370     \or decimoquinto%
3371     \or decimosexto%
3372     \or decimos\'eptimo%
3373     \or decimoctavo%
3374     \or decimonoveno%
3375   \fi
3376}%
3377 \global\let\@@teenthstringspanish\@@teenthstringspanish
```

Hundreds:

```
3378 \newcommand*{\@@hundredthstringspanish[1]}{%
3379   \ifcase#1\relax
3380     \or cent\'esimo%
3381     \or ducent\'esimo%
3382     \or tricent\'esimo%
3383     \or cuadringent\'esimo%
3384     \or quingent\'esimo%
3385     \or sexcent\'esimo%
3386     \or septing\'esimo%
3387     \or octingent\'esimo%
3388     \or noningent\'esimo%
3389   \fi
3390}%
3391 \global\let\@@hundredthstringspanish\@@hundredthstringspanish
```

Units (feminine):

```

3392 \newcommand*\@@unitthstringFspanish[1]{%
3393   \ifcase#1\relax
3394     cera%
3395     \or primera%
3396     \or segunda%
3397     \or tercera%
3398     \or cuarta%
3399     \or quinta%
3400     \or sexta%
3401     \or s\'eptima%
3402     \or octava%
3403     \or novena%
3404   \fi
3405 }%
3406 \global\let\@@unitthstringFspanish\@@unitthstringFspanish

  Tens (feminine):
3407 \newcommand*\@@tenthstringFspanish[1]{%
3408   \ifcase#1\relax
3409     \or d\'ecima%
3410     \or vig\'esima%
3411     \or trig\'esima%
3412     \or cuadrag\'esima%
3413     \or quincuag\'esima%
3414     \or sexag\'esima%
3415     \or septuag\'esima%
3416     \or octog\'esima%
3417     \or nonag\'esima%
3418   \fi
3419 }%
3420 \global\let\@@tenthstringFspanish\@@tenthstringFspanish

  Teens (feminine)
3421 \newcommand*\@@teenthstringFspanish[1]{%
3422   \ifcase#1\relax
3423     d\'ecima%
3424     \or und\'ecima%
3425     \or duod\'ecima%
3426     \or decimotercera%
3427     \or decimocuarta%
3428     \or decimoquinta%
3429     \or decimosexta%
3430     \or decimos\'eptima%
3431     \or decimoctava%
3432     \or decimonovena%
3433   \fi
3434 }%
3435 \global\let\@@teenthstringFspanish\@@teenthstringFspanish

  Hundreds (feminine)
3436 \newcommand*\@@hundredthstringFspanish[1]{%

```

```

3437 \ifcase#1\relax
3438 \or cent\'esima%
3439 \or ducent\'esima%
3440 \or trident\'esima%
3441 \or cuadingent\'esima%
3442 \or quingent\'esima%
3443 \or sexcent\'esima%
3444 \or septing\'esima%
3445 \or octingent\'esima%
3446 \or noningent\'esima%
3447 \fi
3448 }%
3449 \global\let\@@hundredthstringFspanish\@@hundredthstringFspanish

    As above, but with initial letters in upper case
3450 \newcommand*\@@Unitthstringspanish[1]{%
3451 \ifcase#1\relax
3452 Cero%
3453 \or Primero%
3454 \or Segundo%
3455 \or Tercero%
3456 \or Cuarto%
3457 \or Quinto%
3458 \or Sexto%
3459 \or S\'eptimo%
3460 \or Octavo%
3461 \or Noveno%
3462 \fi
3463 }%
3464 \global\let\@@Unitthstringspanish\@@Unitthstringspanish

    Tens:
3465 \newcommand*\@@Tenthstringspanish[1]{%
3466 \ifcase#1\relax
3467 \or D\'ecimo%
3468 \or Vig\'esimo%
3469 \or Trig\'esimo%
3470 \or Cuadrag\'esimo%
3471 \or Quincuag\'esimo%
3472 \or Sexag\'esimo%
3473 \or Septuag\'esimo%
3474 \or Octog\'esimo%
3475 \or Nonag\'esimo%
3476 \fi
3477 }%
3478 \global\let\@@Tenthstringspanish\@@Tenthstringspanish

    Teens:
3479 \newcommand*\@@Teenthstringspanish[1]{%
3480 \ifcase#1\relax
3481 D\'ecimo%

```

```

3482 \or Und\'ecimo%
3483 \or Duod\'ecimo%
3484 \or Decimotercero%
3485 \or Decimocuarto%
3486 \or Decimoquinto%
3487 \or Decimosexto%
3488 \or Decimos\'eptimo%
3489 \or Decimoctavo%
3490 \or Decimonoveno%
3491 \fi
3492 }%
3493 \global\let\@@Teenthstringspanish\@@Teenthstringspanish

Hundreds
3494 \newcommand*\@@Hundredthstringspanish[1]{%
3495 \ifcase#1\relax
3496 \or Cent\'esimo%
3497 \or Ducent\'esimo%
3498 \or Tricent\'esimo%
3499 \or Cuadringent\'esimo%
3500 \or Quingent\'esimo%
3501 \or Sexcent\'esimo%
3502 \or Septing\'esimo%
3503 \or Octingent\'esimo%
3504 \or Noningent\'esimo%
3505 \fi
3506 }%
3507 \global\let\@@Hundredthstringspanish\@@Hundredthstringspanish

As above, but feminine.
3508 \newcommand*\@@UnitthstringFspanish[1]{%
3509 \ifcase#1\relax
3510 Cera%
3511 \or Primera%
3512 \or Segunda%
3513 \or Tercera%
3514 \or Cuarta%
3515 \or Quinta%
3516 \or Sexta%
3517 \or S\'eptima%
3518 \or Octava%
3519 \or Novena%
3520 \fi
3521 }%
3522 \global\let\@@UnitthstringFspanish\@@UnitthstringFspanish

Tens (feminine)
3523 \newcommand*\@@TenthstringFspanish[1]{%
3524 \ifcase#1\relax
3525 \or D\'ecima%
3526 \or Vig\'esima%

```

```

3527 \or Trig\'esima%
3528 \or Cuadrag\'esima%
3529 \or Quincuag\'esima%
3530 \or Sexag\'esima%
3531 \or Septuag\'esima%
3532 \or Octog\'esima%
3533 \or Nonag\'esima%
3534 \fi
3535 }%
3536 \global\let\@@TenthstringFspanish\@@TenthstringFspanish

  Teens (feminine):
3537 \newcommand*\@@TeenthstringFspanish[1]{%
3538 \ifcase#1\relax
3539 D\'ecima%
3540 \or Und\'ecima%
3541 \or Duod\'ecima%
3542 \or Decimotercera%
3543 \or Decimocuarta%
3544 \or Decimoquinta%
3545 \or Decimosexta%
3546 \or Decimos\'eptima%
3547 \or Decimoctava%
3548 \or Decimonovena%
3549 \fi
3550 }%
3551 \global\let\@@TeenthstringFspanish\@@TeenthstringFspanish

  Hundreds (feminine):
3552 \newcommand*\@@HundredthstringFspanish[1]{%
3553 \ifcase#1\relax
3554 \or Cent\'esima%
3555 \or Ducent\'esima%
3556 \or Tricent\'esima%
3557 \or Cuadringent\'esima%
3558 \or Quingent\'esima%
3559 \or Sexcent\'esima%
3560 \or Septing\'esima%
3561 \or Octingent\'esima%
3562 \or Noningent\'esima%
3563 \fi
3564 }%
3565 \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.)
3566 \newcommand*\@@numberstringspanish[2]{%
3567 \ifnum#1>99999

```

```

3568 \PackageError{fmtcount}{Out of range}%
3569 {This macro only works for values less than 100000}%
3570 \else
3571 \ifnum#1<0
3572 \PackageError{fmtcount}{Negative numbers not permitted}%
3573 {This macro does not work for negative numbers, however
3574 you can try typing "minus" first, and then pass the modulus of
3575 this number}%
3576 \fi
3577 \fi
3578 \def#2{}%
3579 \@strctr=#1\relax \divide\@strctr by 1000\relax
3580 \ifnum\@strctr>9
    #1 is greater or equal to 10000
3581 \divide\@strctr by 10
3582 \ifnum\@strctr>1
3583 \let\@@fc@numstr#2\relax
3584 \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
3585 \@strctr=#1 \divide\@strctr by 1000\relax
3586 \@FCmodulo{\@strctr}{10}%
3587 \ifnum\@strctr>0\relax
3588 \let\@@fc@numstr#2\relax
3589 \edef#2{\@@fc@numstr\ \@andname\ \@unitstring{\@strctr}}%
3590 \fi
3591 \else
3592 \@strctr=#1\relax
3593 \divide\@strctr by 1000\relax
3594 \@FCmodulo{\@strctr}{10}%
3595 \let\@@fc@numstr#2\relax
3596 \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
3597 \fi
3598 \let\@@fc@numstr#2\relax
3599 \edef#2{\@@fc@numstr\ \@thousand}%
3600 \else
3601 \ifnum\@strctr>0\relax
3602 \ifnum\@strctr>1\relax
3603 \let\@@fc@numstr#2\relax
3604 \edef#2{\@@fc@numstr\@unitstring{\@strctr}\ }%
3605 \fi
3606 \let\@@fc@numstr#2\relax
3607 \edef#2{\@@fc@numstr\@thousand}%
3608 \fi
3609 \fi
3610 \@strctr=#1\relax \@FCmodulo{\@strctr}{1000}%
3611 \divide\@strctr by 100\relax
3612 \ifnum\@strctr>0\relax
3613 \ifnum#1>1000\relax
3614 \let\@@fc@numstr#2\relax
3615 \edef#2{\@@fc@numstr\ }%

```



```

3616 \fi
3617 \@tmpstrctr=#1\relax
3618 \@FCmodulo{\@tmpstrctr}{1000}%
3619 \ifnum\@tmpstrctr=100\relax
3620   \let\@@fc@numstr#2\relax
3621   \edef#2{\@@fc@numstr\@tenstring{10}}%
3622 \else
3623   \let\@@fc@numstr#2\relax
3624   \edef#2{\@@fc@numstr\@hundredstring{\@strctr}}%
3625 \fi
3626 \fi
3627 \@strctr=#1\relax \@FCmodulo{\@strctr}{100}%
3628 \ifnum#1>100\relax
3629   \ifnum\@strctr>0\relax
3630     \let\@@fc@numstr#2\relax
3631     \edef#2{\@@fc@numstr\ }%
3632   \fi
3633 \fi
3634 \ifnum\@strctr>29\relax
3635   \divide\@strctr by 10\relax
3636   \let\@@fc@numstr#2\relax
3637   \edef#2{\@@fc@numstr\@tenstring{\@strctr}}%
3638   \@strctr=#1\relax \@FCmodulo{\@strctr}{10}%
3639   \ifnum\@strctr>0\relax
3640     \let\@@fc@numstr#2\relax
3641     \edef#2{\@@fc@numstr\ \@andname\ \@unitstring{\@strctr}}%
3642   \fi
3643 \else
3644   \ifnum\@strctr<10\relax
3645     \ifnum\@strctr=0\relax
3646       \ifnum#1<100\relax
3647         \let\@@fc@numstr#2\relax
3648         \edef#2{\@@fc@numstr\@unitstring{\@strctr}}%
3649       \fi
3650     \else
3651       \let\@@fc@numstr#2\relax
3652       \edef#2{\@@fc@numstr\@unitstring{\@strctr}}%
3653     \fi
3654   \else
3655     \ifnum\@strctr>19\relax
3656       \@FCmodulo{\@strctr}{10}%
3657       \let\@@fc@numstr#2\relax
3658       \edef#2{\@@fc@numstr\@twentystring{\@strctr}}%
3659     \else
3660       \@FCmodulo{\@strctr}{10}%
3661       \let\@@fc@numstr#2\relax
3662       \edef#2{\@@fc@numstr\@teenstring{\@strctr}}%
3663     \fi
3664   \fi

```

```

3665 \fi
3666 }%
3667 \global\let\@@numberstringspanish\@@numberstringspanish

```

As above, but for ordinals

```

3668 \newcommand*\@@ordinalstringspanish[2]{%
3669 \@strctr=#1\relax
3670 \ifnum#1>99999
3671 \PackageError{fmtcount}{Out of range}%
3672 {This macro only works for values less than 100000}%
3673 \else
3674 \ifnum#1<0
3675 \PackageError{fmtcount}{Negative numbers not permitted}%
3676 {This macro does not work for negative numbers, however
3677 you can try typing "minus" first, and then pass the modulus of
3678 this number}%
3679 \else
3680 \def#2{}%
3681 \ifnum\@strctr>999\relax
3682 \divide\@strctr by 1000\relax
3683 \ifnum\@strctr>1\relax
3684 \ifnum\@strctr>9\relax
3685 \@tmpstrctr=\@strctr
3686 \ifnum\@strctr<20
3687 \@FCmodulo{\@tmpstrctr}{10}%
3688 \let\@@fc@ordstr#2\relax
3689 \edef#2{\@@fc@ordstr\@teenthstring{\@tmpstrctr}}%
3690 \else
3691 \divide\@tmpstrctr by 10\relax
3692 \let\@@fc@ordstr#2\relax
3693 \edef#2{\@@fc@ordstr\@tenthstring{\@tmpstrctr}}%
3694 \@tmpstrctr=\@strctr
3695 \@FCmodulo{\@tmpstrctr}{10}%
3696 \ifnum\@tmpstrctr>0\relax
3697 \let\@@fc@ordstr#2\relax
3698 \edef#2{\@@fc@ordstr\@unitthstring{\@tmpstrctr}}%
3699 \fi
3700 \fi
3701 \else
3702 \let\@@fc@ordstr#2\relax
3703 \edef#2{\@@fc@ordstr\@unitstring{\@strctr}}%
3704 \fi
3705 \fi
3706 \let\@@fc@ordstr#2\relax
3707 \edef#2{\@@fc@ordstr\@thousandth}%
3708 \fi
3709 \@strctr=#1\relax
3710 \@FCmodulo{\@strctr}{1000}%
3711 \ifnum\@strctr>99\relax
3712 \@tmpstrctr=\@strctr

```

```

3713 \divide\@tmpstrctr by 100\relax
3714 \ifnum#1>1000\relax
3715     \let\@fc@ordstr#2\relax
3716     \edef#2{\@fc@ordstr\ }%
3717 \fi
3718 \let\@fc@ordstr#2\relax
3719 \edef#2{\@fc@ordstr\@hundredthstring{\@tmpstrctr}}%
3720 \fi
3721 \@FCmodulo{\@strctr}{100}%
3722 \ifnum#1>99\relax
3723     \ifnum\@strctr>0\relax
3724         \let\@fc@ordstr#2\relax
3725         \edef#2{\@fc@ordstr\ }%
3726     \fi
3727 \fi
3728 \ifnum\@strctr>19\relax
3729     \@tmpstrctr=\@strctr
3730     \divide\@tmpstrctr by 10\relax
3731     \let\@fc@ordstr#2\relax
3732     \edef#2{\@fc@ordstr\@tenthstring{\@tmpstrctr}}%
3733     \@tmpstrctr=\@strctr
3734     \@FCmodulo{\@tmpstrctr}{10}%
3735     \ifnum\@tmpstrctr>0\relax
3736         \let\@fc@ordstr#2\relax
3737         \edef#2{\@fc@ordstr\ \@unitthstring{\@tmpstrctr}}%
3738     \fi
3739 \else
3740     \ifnum\@strctr>9\relax
3741         \@FCmodulo{\@strctr}{10}%
3742         \let\@fc@ordstr#2\relax
3743         \edef#2{\@fc@ordstr\@teenthstring{\@strctr}}%
3744     \else
3745         \ifnum\@strctr=0\relax
3746             \ifnum#1=0\relax
3747                 \let\@fc@ordstr#2\relax
3748                 \edef#2{\@fc@ordstr\@unitstring{0}}%
3749             \fi
3750         \else
3751             \let\@fc@ordstr#2\relax
3752             \edef#2{\@fc@ordstr\@unitthstring{\@strctr}}%
3753         \fi
3754     \fi
3755 \fi
3756 \fi
3757 \fi
3758 }%
3759 \global\let\@@ordinalstringspanish\@@ordinalstringspanish

```

10.1.16 fc-UKenglish.def

English definitions

```
3760 \ProvidesFCLanguage{UKenglish}[2013/08/17]%
```

Loaded fc-english.def if not already loaded

```
3761 \FCloadlang{english}%
```

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

```
3762 \global\let\@ordinalMUKenglish\@ordinalMenglish
3763 \global\let\@ordinalFUKenglish\@ordinalMenglish
3764 \global\let\@ordinalNUKenglish\@ordinalMenglish
3765 \global\let\@numberstringMUKenglish\@numberstringMenglish
3766 \global\let\@numberstringFUKenglish\@numberstringMenglish
3767 \global\let\@numberstringNUKenglish\@numberstringMenglish
3768 \global\let\@NumberstringMUKenglish\@NumberstringMenglish
3769 \global\let\@NumberstringFUKenglish\@NumberstringMenglish
3770 \global\let\@NumberstringNUKenglish\@NumberstringMenglish
3771 \global\let\@ordinalstringMUKenglish\@ordinalstringMenglish
3772 \global\let\@ordinalstringFUKenglish\@ordinalstringMenglish
3773 \global\let\@ordinalstringNUKenglish\@ordinalstringMenglish
3774 \global\let\@OrdinalstringMUKenglish\@OrdinalstringMenglish
3775 \global\let\@OrdinalstringFUKenglish\@OrdinalstringMenglish
3776 \global\let\@OrdinalstringNUKenglish\@OrdinalstringMenglish
```

10.1.17 fc-USenglish.def

US English definitions

```
3777 \ProvidesFCLanguage{USenglish}[2013/08/17]%
```

Loaded fc-english.def if not already loaded

```
3778 \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.)

```
3779 \global\let\@ordinalMUSenglish\@ordinalMenglish
3780 \global\let\@ordinalFUSenglish\@ordinalMenglish
3781 \global\let\@ordinalNUSenglish\@ordinalMenglish
3782 \global\let\@numberstringMUSenglish\@numberstringMenglish
3783 \global\let\@numberstringFUSenglish\@numberstringMenglish
3784 \global\let\@numberstringNUSenglish\@numberstringMenglish
3785 \global\let\@NumberstringMUSenglish\@NumberstringMenglish
3786 \global\let\@NumberstringFUSenglish\@NumberstringMenglish
3787 \global\let\@NumberstringNUSenglish\@NumberstringMenglish
3788 \global\let\@ordinalstringMUSenglish\@ordinalstringMenglish
3789 \global\let\@ordinalstringFUSenglish\@ordinalstringMenglish
3790 \global\let\@ordinalstringNUSenglish\@ordinalstringMenglish
3791 \global\let\@OrdinalstringMUSenglish\@OrdinalstringMenglish
3792 \global\let\@OrdinalstringFUSenglish\@OrdinalstringMenglish
3793 \global\let\@OrdinalstringNUSenglish\@OrdinalstringMenglish
```

10.2 fcnumparser.sty

```
3794 \NeedsTeXFormat{LaTeX2e}
3795 \ProvidesPackage{fcnumparser}[2017/06/15]
```

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

```
3796 \def\fc@counter@parser#1{%
3797   \expandafter\fc@number@parser\expandafter{\the#1.}%
3798 }
3799 \newcount\fc@digit@counter
3800
3801 \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.

```
3802 \def\fc@number@analysis#1\fc@nil{%
```

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

```
3803   \def\@tempb##1.##2\fc@nil{\def\fc@integer@part{##1}\def\@tempa{##2}}%
3804   \@tempb#1.\fc@end\fc@nil
3805   \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.

```
3806   \def\@tempb##1,##2\fc@nil{\def\fc@integer@part{##1}\def\@tempa{##2}}%
3807   \@tempb#1,\fc@end\fc@nil
3808   \ifx\@tempa\fc@end@
```

No comma either, so fractional part is set empty.

```
3809   \def\fc@fractional@part{}%
3810   \else
```

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

```
3811   \def\@tempb##1,\fc@end{\def\fc@fractional@part{##1}}%
3812   \expandafter\@tempb\@tempa
3813   \fi
3814   \else
```

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

```
3815   \def\@tempb##1.\fc@end{\def\fc@fractional@part{##1}}%
3816   \expandafter\@tempb\@tempa
3817   \fi
3818 }
```

c@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>.

```
3819 \def\fc@number@parser#1{%
```

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

```

3820 \let\@tempa\@empty
3821 \def\@tempb##1##2\fc@nil{%
3822   \def\@tempc{##1}%
3823   \ifx\@tempc\space
3824   \else
3825     \expandafter\def\expandafter\@tempa\expandafter{\@tempa ##1}%
3826   \fi
3827   \def\@tempc{##2}%
3828   \ifx\@tempc\@empty
3829     \expandafter\@gobble
3830   \else
3831     \expandafter\@tempb
3832   \fi
3833   ##2\fc@nil
3834 }%
3835 \@tempb#1\fc@nil

```

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

```

3836 \def\@tempb##1##2\fc@nil{\def\fc@sign{##1}\def\fc@number{##2}}%
3837 \expandafter\@tempb\@tempa\fc@nil
3838 \expandafter\if\fc@sign+%
3839   \def\fc@sign@case{1}%
3840 \else
3841   \expandafter\if\fc@sign-%
3842     \def\fc@sign@case{2}%
3843   \else
3844     \def\fc@sign{}%
3845     \def\fc@sign@case{0}%
3846     \let\fc@number\@tempa
3847   \fi
3848 \fi
3849 \ifx\fc@number\@empty
3850   \PackageError{fcnumparser}{Invalid number}{Number must contain at least one non blank
3851     character after sign}%
3852 \fi

```

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

```

3853 \expandafter\fc@number@analysis\fc@number\fc@nil

```

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

```

3854 \expandafter\fc@digit@counter\fc@unit@weight
3855 \expandafter\fc@parse@integer@digits\fc@integer@part\fc@end\fc@nil

```

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

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

```

3856 \fc@digit@counter - \fc@digit@counter
3857 \advance\fc@digit@counter by \fc@unit@weight
3858 \advance\fc@digit@counter by \fc@unit@weight
3859 \advance\fc@digit@counter by -1 %
3860 \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 $\text{\fc@digit@}\langle i + \text{offset} \rangle$ to $\text{\fc@digit@}\langle i \rangle$. First we compute offset into \@tempi .

```

3861 {%
3862   \count0 \fc@unit@weight\relax
3863   \count1 \fc@max@weight\relax
3864   \advance\count0 by -\count1 %
3865   \advance\count0 by -1 %
3866   \def\@tempa##1{\def\@tempb{\def\@tempi{##1}}}%
3867   \expandafter\@tempa\expandafter{\the\count0}%
3868   \expandafter
3869 } \@tempb

```

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

```

3870 \expandafter\fc@digit@counter\fc@unit@weight
3871 \def\@templ{%
3872   \ifnum\fc@digit@counter>\fc@max@weight
3873     \let\next\relax
3874   \else

```

Here is the loop body:

```

3875   {%
3876     \count0 \@tempi
3877     \advance\count0 by \fc@digit@counter
3878     \expandafter\def\expandafter\@tempd\expandafter{\csname fc@digit@the\count0\endcsname}
3879     \expandafter\def\expandafter\@tempe\expandafter{\csname fc@digit@the\fc@digit@counter\endcsname}
3880     \def\@tempa###1###2{\def\@tempb{\let###1###2}}%
3881     \expandafter\expandafter\expandafter\@tempa\expandafter\@tempe\expandafter\@tempd
3882     \expandafter
3883   } \@tempb
3884   \advance\fc@digit@counter by 1 %
3885   \fi
3886   \next
3887 }%
3888 \let\next\@templ
3889 \@templ

```

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

```

3890 \expandafter\fc@digit@counter\fc@unit@weight
3891 \expandafter\fc@parse@integer@digits\fc@fractional@part\fc@end\fc@nil
3892 \edef\fc@min@weight{\the\fc@digit@counter}%
3893 }

```

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

```

3894 \ifcsundef\fc@parse@integer@digits\{}{}%

```

```

3895 \PackageError{fcnumparser}{Duplicate definition}{Redefinition of
3896   macro 'fc@parse@integer@digits'}}
3897 \def\fc@parse@integer@digits#1#2\fc@nil{%
3898   \def\@tempa{#1}%
3899   \ifx\@tempa\fc@end@
3900     \def\next##1\fc@nil{%
3901       \else
3902       \let\next\fc@parse@integer@digits
3903       \advance\fc@digit@counter by -1
3904       \expandafter\def\csname fc@digit@\the\fc@digit@counter\endcsname{#1}%
3905       \fi
3906       \next#2\fc@nil
3907 }
3908
3909
3910 \newcommand*{\fc@unit@weight}{0}
3911

```

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.

```

3912 \ifcsundef{fc@read@unit}{%
3913   \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.

```

3914 \def\fc@read@unit#1#2{%
3915   \ifnum#2>\fc@max@weight
3916     #1=0\relax
3917   \else
3918     \ifnum#2<\fc@min@weight
3919       #1=0\relax
3920     \else
3921       {%
3922         \edef\@tempa{\number#2}%
3923         \count0=\@tempa
3924         \edef\@tempa{\csname fc@digit@\the\count0\endcsname}%
3925         \def\@tempb##1{\def\@tempa{#1=##1\relax}}%
3926         \expandafter\@tempb\expandafter{\@tempa}%
3927         \expandafter
3928       }\@tempa
3929     \fi
3930   \fi
3931 }

```

`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.

```
3932 \ifcsundef{fc@read@hundred}{%}{}%
3933 \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

```
3934 \def\fc@read@hundred#1#2{%
3935   {%
3936     \fc@read@unit{\count0}{#2}%
3937     \def\@tempa#1{\fc@read@unit{\count1}{##1}}%
3938     \count2=#2%
3939     \advance\count2 by 1 %
3940     \expandafter\@tempa{\the\count2}%
3941     \multiply\count1 by 10 %
3942     \advance\count1 by \count0 %
3943     \def\@tempa#1{\def\@tempb{#1=##1\relax}}
3944     \expandafter\@tempa\expandafter{\the\count1}%
3945     \expandafter
3946   }\@tempb
3947 }
```

`\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.

```
3948 \ifcsundef{fc@read@thousand}{%}{}%
3949 \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro
3950   '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

```
3951 \def\fc@read@thousand#1#2{%
3952   {%
3953     \fc@read@unit{\count0}{#2}%
3954     \def\@tempa#1{\fc@read@hundred{\count1}{##1}}%
3955     \count2=#2%
3956     \advance\count2 by 1 %
3957     \expandafter\@tempa{\the\count2}%
3958     \multiply\count1 by 10 %
3959     \advance\count1 by \count0 %
3960     \def\@tempa#1{\def\@tempb{#1=##1\relax}}
3961     \expandafter\@tempa\expandafter{\the\count1}%
3962     \expandafter
3963   }\@tempb
3964 }
```

`\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.

```
3965 \ifcsundef{fc@read@myriad}{%}{}%
3966 \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro
```

```
3967   '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

```
3968 \def\fc@read@myriad#1#2{%
```

```
3969   {%
```

```
3970     \fc@read@hundred{\count0}{#2}%
```

```
3971     \def\@tempa##1{\fc@read@hundred{\count1}{##1}}%
```

```
3972     \count2=#2
```

```
3973     \advance\count2 by 2
```

```
3974     \expandafter\@tempa{\the\count2}%
```

```
3975     \multiply\count1 by 100 %
```

```
3976     \advance\count1 by \count0 %
```

```
3977     \def\@tempa##1{\def\@tempb{#1=##1\relax}}%
```

```
3978     \expandafter\@tempa\expandafter{\the\count1}%
```

```
3979     \expandafter
```

```
3980   }\@tempb
```

```
3981 }
```

`\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.

```
3982 \ifcsundef{fc@check@nonzeros}{}{%
```

```
3983   \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro
```

```
3984     'fc@check@nonzeros'}}}
```

Arguments as follows:

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

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

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

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

```
3985 \def\fc@check@nonzeros#1#2#3{%
```

```
3986   {%
```

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

```
3987     \edef\@tempa{\number#1}%
```

```
3988     \edef\@tempb{\number#2}%
```

```
3989     \count0=\@tempa
```

```
3990     \count1=\@tempb\relax
```

Then we do the real job

```
3991   \fc@@check@nonzeros@inner
```

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

```
3992   \def\@tempd##1{\def\@tempa{\def#3{##1}}}%
```

```
3993   \expandafter\@tempd\expandafter{\@tempc}%
```

```
3994   \expandafter
```

```
3995   }\@tempa
```

```

3996 }
\@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.
3997 \def\fc@@check@nonzeros@inner{%
3998   \ifnum\count0<\fc@min@weight
3999     \count0=\fc@min@weight\relax
4000   \fi
4001   \ifnum\count1>\fc@max@weight\relax
4002     \count1=\fc@max@weight
4003   \fi
4004   \count2\count0 %
4005   \advance\count2 by 1 %
4006   \ifnum\count0>\count1 %
4007     \PackageError{fcnumparser}{Unexpected arguments}{Number in argument 2 of macro
4008       'fc@check@nonzeros' must be at least equal to number in argument 1}%
4009   \else
4010     \fc@@check@nonzeros@inner@loopbody
4011     \ifnum\@tempc>0 %
4012       \ifnum\@tempc<9 %
4013         \ifnum\count0>\count1 %
4014           \else
4015             \let\@tempd\@tempc
4016             \fc@@check@nonzeros@inner@loopbody
4017             \ifnum\@tempc=0 %
4018               \let\@tempc\@tempd
4019             \else
4020               \def\@tempc{9}%
4021             \fi
4022           \fi
4023         \fi
4024       \fi
4025     \fi
4026 }
4027 \def\fc@@check@nonzeros@inner@loopbody{%
4028   % \@tempc <- digit of weight \count0
4029   \expandafter\let\expandafter\@tempc\csname fc@digit@the\count0\endcsname
4030   \advance\count0 by 1 %
4031   \ifnum\@tempc=0 %
4032     \ifnum\count0>\count1 %

```

```

4033     \let\next\relax
4034   \else
4035     \let\next\fc@@check@nonzeros@inner@loopbody
4036   \fi
4037 \else
4038   \ifnum\count0>\count2 %
4039     \def\@tempc{9}%
4040   \fi
4041   \let\next\relax
4042 \fi
4043 \next
4044 }

```

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

```

4045 \ifcsundef{fc@intpart@find@last}{\}%
4046 \PackageError{fcnumparser}{Duplicate definition}{Redefinition of macro
4047   'fc@intpart@find@last'}}

```

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

```

4048 \def\fc@intpart@find@last#1{%
4049   {%

```

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

```

4050   \count0=\fc@unit@weight\space
4051   \ifnum\count0<\fc@min@weight\space
4052     \count0=\fc@min@weight\space
4053   \fi

```

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

```

4054   \def\@templ{%
4055     \ifnum\curname fc@digit@the\count0\endcsname=0 %
4056     \advance\count0 by 1 %
4057     \ifnum\count0>\fc@max@weight\space
4058       \let\next\relax
4059     \fi
4060   \else
4061     \let\next\relax
4062   \fi
4063   \next
4064 }%
4065 \let\next\@templ
4066 \@templ

```

Now propagate result after closing bracket into counter #1.

```

4067   \toks0{#1}%
4068   \edef\@tempa{\the\toks0=\the\count0}%
4069   \expandafter

```

```

4070 } \@tempa\space
4071 }

```

`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

```

4072 \ifcsundef{fc@get@last@word}{ }\PackageError{fcnumparser}{Duplicate definition}{Redefinition
4073   of macro 'fc@get@last@word'}}}%
4074 \def\fc@get@last@word#1#2#3{%
4075   {%

```

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`.

```

4076   \def\@tempa##1\fc@wcase##2\@nil\fc@end{%
4077     \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`.

```

4078     \toks1{##2\fc@wcase}%
4079   }%
4080   \@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`.

```

4081   \def\@tempa##1\fc@wcase##2\fc@end{%
4082     \toks2{##1}%
4083     \def\@tempb{##2}%
4084     \toks3{##2}%
4085   }%

```

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

```

4086   \toksdef\@tempt0 %

```

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

```

4087   \def\@templ{%
4088     \expandafter\@tempa\the\toks1 \fc@end
4089     \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.

```

4090     \let\next\relax
4091   \else

```

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

```

4092     \expandafter\expandafter\expandafter\@tempt
4093     \expandafter\expandafter\expandafter{%
4094       \expandafter\the\expandafter\@tempt
4095       \expandafter\fc@wcase\the\toks2}%
4096     \toks1\toks3 %

```

```

4097     \fi
4098     \next
4099   }%
4100   \let\next\@templ
4101   \@templ
4102   \edef\@tempa{\def\noexpand#2{\the\toks0}\def\noexpand#3{\the\toks2}}%
4103   \expandafter
4104   }\@tempa
4105 }

```

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

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

```

4106 \ifcsundef{fc@get@last@letter}{\PackageError{fcnumparser}{Duplicate definition}{Redefinition
4107   of macro 'fc@get@last@letter'}}%
4108 \def\fc@get@last@letter#1#2#3{%
4109   {%

```

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`.

```

4110   \toks1{#1}%
4111   \toks0{}%
4112   \toksdef\@tempt0 %

```

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

```

4113   \def\@tempa##1##2\fc@nil{%
4114     \toks2{##1}%
4115     \toks3{##2}%
4116     \def\@tempb{##2}%
4117   }%

```

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

```

4118   \def\@templ{%
4119     \expandafter\@tempa\the\toks1 \fc@nil
4120     \ifx\@tempb\@empty

```

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

```

4121     \let\next\relax
4122   \else

```

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

```

4123     \expandafter\expandafter\expandafter\@tempt
4124     \expandafter\expandafter\expandafter{%
4125     \expandafter\the\expandafter\@tempt
4126     \the\toks2}%

```

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

```

4127     \toks1\toks3 %
4128     \fi
4129     \next
4130   }%

```

Here run the loop.

```
4131 \let\next\@temp1
4132 \next
```

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

```
4133 \edef\@tempa{\def\noexpand#2{\the\toks0}\def\noexpand#3{\the\toks1}}%
4134 \expandafter
4135 }\@tempa
4136 }%
```

10.3 fcprefix.sty

Pseudo-latin prefixes.

```
4137 \NeedsTeXFormat{LaTeX2e}
4138 \ProvidesPackage{fcprefix}[2012/09/28]
4139 \RequirePackage{ifthen}
4140 \RequirePackage{keyval}
4141 \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’.

```
4142 \define@key{fcprefix}{use duode and unde}[below20]{%
4143 \ifthenelse{\equal{#1}{below20}}{%
4144 \def\fc@duodeandunde{2}%
4145 }{%
4146 \ifthenelse{\equal{#1}{never}}{%
4147 \def\fc@duodeandunde{0}%
4148 }{%
4149 \PackageError{fcprefix}{Unexpected option}{%
4150 Option ‘use duode and unde’ expects ‘below 20’ or ‘never’ }%
4151 }%
4152 }%
4153 }
```

Default is ‘below 20’ like in French.

```
4154 \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.

```
4155 \define@key{fcprefix}{numeral u in duo}[false]{%
4156 \ifthenelse{\equal{#1}{false}}{%
4157 \let\fc@u@in@duo\@empty
4158 }{%
4159 \ifthenelse{\equal{#1}{true}}{%
4160 \def\fc@u@in@duo{u}%
4161 }{%
4162 \PackageError{fcprefix}{Unexpected option}{%
4163 Option ‘numeral u in duo’ expects ‘true’ or ‘false’ }%
4164 }%
4165 }%
4166 }
```

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.

```

4167 \define@key{fcprefix}{e accute}[false]{%
4168   \ifthenelse{\equal{#1}{false}}{%
4169     \let\fc@prefix@eaccute\@firstofone
4170   }{%
4171     \ifthenelse{\equal{#1}{true}}{%
4172       \let\fc@prefix@eaccute\'%
4173     }{%
4174       \PackageError{fcprefix}{Unexpected option}{%
4175         Option ‘e accute’ expects ‘true’ or ‘false’ }%
4176     }%
4177   }%
4178 }

```

Default is to set accute accent like in French.

```

4179 \let\fc@prefix@eaccute\'%

```

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

recursive for which millia squared is noted as ‘milliamillia’

arabic for which millia squared is noted as ‘millia²’

prefix for which millia squared is noted as ‘bismillia’

```

4180 \define@key{fcprefix}{power of millia}[prefix]{%
4181   \ifthenelse{\equal{#1}{prefix}}{%
4182     \let\fc@power@of@millia@init\@gobbletwo
4183     \let\fc@power@of@millia\fc@@prefix@millia
4184   }{%
4185     \ifthenelse{\equal{#1}{arabic}}{%
4186       \let\fc@power@of@millia@init\@gobbletwo
4187       \let\fc@power@of@millia\fc@@arabic@millia
4188     }{%
4189       \ifthenelse{\equal{#1}{recursive}}{%
4190         \let\fc@power@of@millia@init\fc@@recurse@millia@init
4191         \let\fc@power@of@millia\fc@@recurse@millia
4192       }{%
4193         \PackageError{fcprefix}{Unexpected option}{%
4194           Option ‘power of millia’ expects ‘recursive’, ‘arabic’, or ‘prefix’ }%
4195       }%
4196     }%
4197   }%
4198 }

```

Arguments as follows:

#1 output macro

#2 number with current weight w

```

4199 \def\fc@@recurse@millia#1#2{%
4200   \let\@temp#1%
4201   \edef#1{millia\@temp}%
4202 }

```


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

#1 output macro

#2 number with current weight w

```
4203 \def\fc@@recurse@millia@init#1#2{%
```

```
4204  {%
```

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

```
4205  \edef\@tempb{\number#2}%
```

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

```
4206  \count0=0 %
```

```
4207  \let\@tempa\@empty
```

```
4208  \loop
```

```
4209    \ifnum\count0<\@tempb
```

```
4210      \advance\count0 by 1 %
```

```
4211      \expandafter\def
```

```
4212        \expandafter\@tempa\expandafter{\@tempa millia}%
```

```
4213  \repeat
```

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

```
4214  \edef\@tempb{\def\noexpand#1{\@tempa}}%
```

```
4215  \expandafter
```

```
4216  }\@tempb
```

```
4217 }
```

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

#1 output macro

#2 number with current weight w

```
4218 \def\fc@@arabic@millia#1#2{%
```

```
4219  \ifnum#2=0 %
```

```
4220    \let#1\@empty
```

```
4221  \else
```

```
4222    \edef#1{millia\^{}\the#2}%
```

```
4223  \fi
```

```
4224 }
```

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

#1 output macro

#2 number with current weight w

```
4225 \def\fc@@prefix@millia#1#2{%
```

```
4226  \fc@@latin@numeral@pefix{#2}{#1}%
```

```
4227 }
```

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

```
4228 \let\fc@power@of@millia@init\@gobbletwo
```

```
4229 \let\fc@power@of@millia\fc@@prefix@millia
```

`\fc@@latin@cardinal@pefix` 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.

```
4230 \ifcsundef\fc@@latin@cardinal@pefix}{%}
```

```
4231  \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

```
4232 \def\fc@@latin@cardinal@pefix#1#2{%
```

```
4233  {%
```

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

```
4234 \edef\@tempa{\number#1}%
```

Now parse number from expanded input.

```
4235 \expandafter\fc@number@parser\expandafter{\@tempa}%
```

```
4236 \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 \@tempi.

```
4237 \let\@tempt\@empty
```

```
4238 \def\@tempu{t}%
```

\@tempm will hold the millia^{n÷3}

```
4239 \let\@tempm\@empty
```

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

```
4240 \def\@templ{%
```

```
4241 \ifnum\count2>\fc@max@weight
```

```
4242 \let\next\relax
```

```
4243 \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.

```
4244 \fc@read@unit{\count3}{\count2}%
```

```
4245 \advance\count2 by 1 %
```

```
4246 \fc@read@unit{\count4}{\count2}%
```

```
4247 \advance\count2 by 1 %
```

```
4248 \fc@read@unit{\count5}{\count2}%
```

```
4249 \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.

```
4250 \edef\@tempn{%
```

```
4251 \ifnum\count3=0\else 1\fi
```

```
4252 \ifnum\count4=0\else 1\fi
```

```
4253 \ifnum\count5=0\else 1\fi
```

```
4254 }%
```

```
4255 \ifx\@tempn\@empty\else
```

```
4256 \let\@tempt\@tempu
```

```
4257 \let\@tempu\@empty
```

```
4258 \fi
```

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

```
4259 \let\@temp\@tempa
```

```
4260 \edef\@tempa{%
```

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

```

4261      \ifcase\count5 %
4262      \or cen%
4263      \or ducen%
4264      \or trecen%
4265      \or quadringen%
4266      \or quingen%
4267      \or sescen%
4268      \or septigen%
4269      \or octingen%
4270      \or nongen%
4271      \fi

```

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

```

4272      \ifnum\count4=0 %
4273      % x0(0..9)
4274      \ifnum\count2=3 %
4275      % Absolute weight zero
4276      \ifcase\count3 \@tempt
4277      \or m%
4278      \or b%
4279      \or tr%
4280      \or quadr%
4281      \or quin\@tempt
4282      \or sex\@tempt
4283      \or sep\@tempt
4284      \or oc\@tempt
4285      \or non%
4286      \fi
4287      \else

```

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

```

4288      \ifcase\count3 %
4289      \or \ifnum\count2>\fc@max@weight\else un\fi
4290      \or d\fc@u@in@duo o%
4291      \or tre%
4292      \or quattuor%
4293      \or quin%
4294      \or sex%
4295      \or septen%
4296      \or octo%
4297      \or novem%
4298      \fi
4299      \fi
4300      \else
4301      % x(10..99)
4302      \ifcase\count3 %
4303      \or un%
4304      \or d\fc@u@in@duo o%
4305      \or tre%

```

```

4306          \or quattuor%
4307          \or quin%
4308          \or sex%
4309          \or septen%
4310          \or octo%
4311          \or novem%
4312          \fi
4313          \ifcase\count4 %
4314          \or dec%
4315          \or vigin\@tempt
4316          \or trigin\@tempt
4317          \or quadragin\@tempt
4318          \or quinquagin\@tempt
4319          \or sexagin\@tempt
4320          \or septuagin\@tempt
4321          \or octogin\@tempt
4322          \or nonagin\@tempt
4323          \fi
4324          \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.

```

4325          \@tempm

```

And append previous version of \@tempa.

```

4326          \@tempp
4327      }%

```

“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’.

```

4328          \fc@power@of@millia\@tempm{\count2}%
4329          \fi
4330          \next
4331      }%
4332      \let\@tempa\@empty
4333      \let\@next\@templ
4334      \@templ

```

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

```

4335      \def\@tempb##1{\def\@tempa{\def#2{##1}}}%
4336      \expandafter\@tempb\expandafter{\@tempa}%
4337      \expandafter
4338      }\@tempa
4339  }

```

`@@latin@numeral@p` 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.

```

4340 \ifcsundef{fc@@latin@numeral@pfix}{\}%
4341 \PackageError{fmtcount}{Duplicate definition}{Redefinition of macro

```

4342 'fc@@latin@numeral@pexif'}}}

Arguments as follows:

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

```
4343 \def\fc@@latin@numeral@pexif#1#2{%
4344   {%
4345     \edef\@tempa{\number#1}%
4346     \def\fc@unit@weight{0}%
4347     \expandafter\fc@number@parser\expandafter{\@tempa}%
4348     \count2=0 %
```

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

```
4349   \let\@tempm\@empty
```

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

```
4350   \def\@templ{%
4351     \ifnum\count2>\fc@max@weight
4352       \let\next\relax
4353     \else
```

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

```
4354       \fc@read@unit{\count3}{\count2}%
4355       \advance\count2 by 1 %
4356       \fc@read@unit{\count4}{\count2}%
4357       \advance\count2 by 1 %
4358       \fc@read@unit{\count5}{\count2}%
4359       \advance\count2 by 1 %
```

Check the use of duodevicies instead of octodecies.

```
4360       \let\@tempn\@secondoftwo
4361       \ifnum\count3>7 %
4362         \ifnum\count4<\fc@duodeandunde
4363           \ifnum\count4>0 %
4364             \let\@tempn\@firstoftwo
4365           \fi
4366         \fi
4367       \fi
4368       \@tempn
4369       {% use duodevicies for eighteen
4370         \advance\count4 by 1 %
4371         \let\@temps\@secondoftwo
4372       }{% do not use duodevicies for eighteen
4373         \let\@temps\@firstoftwo
4374       }%
4375       \let\@tempp\@tempa
4376       \edef\@tempa{%
4377         % hundreds
4378         \ifcase\count5 %
4379         \expandafter\@gobble
```

```

4380         \or c%
4381         \or duc%
4382         \or trec%
4383         \or quadring%
4384         \or quing%
4385         \or sesc%
4386         \or septing%
4387         \or octing%
4388         \or nong%
4389         \fi
4390         {enties}%
4391         \ifnum\count4=0 %

```

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

```

4392         \ifcase\count3 %
4393         \or
4394         \ifnum\count2=3 %
4395             s\fc@prefix@eacute emel%
4396         \else
4397             \ifnum\count2>\fc@max@weight\else un\fi
4398         \fi
4399         \or bis%
4400         \or ter%
4401         \or quater%
4402         \or quinquies%
4403         \or sexies%
4404         \or septies%
4405         \or octies%
4406         \or novies%
4407         \fi
4408         \else

```

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

```

4409         \ifcase\count3 %
4410         \or un%
4411         \or d\fc@u@in@duo o%
4412         \or ter%
4413         \or quater%
4414         \or quin%
4415         \or sex%
4416         \or septen%
4417         \or \@temps{octo}{duod\fc@prefix@eacute e}% x8 = two before next (x+1)0
4418         \or \@temps{novem}{und\fc@prefix@eacute e}% x9 = one before next (x+1)0
4419         \fi
4420         \ifcase\count4 %
4421         % can't get here
4422         \or d\fc@prefix@eacute ec%
4423         \or vic%
4424         \or tric%
4425         \or quadrag%

```

```

4426         \or quinquag%
4427         \or sexag%
4428         \or septuag%
4429         \or octog%
4430         \or nonag%
4431         \fi
4432         ies%
4433     \fi
4434     % Insert the millies(n/3) only if one of \count3 \count4 \count5 is non zero
4435     \@tempm
4436     % add up previous version of \@tempa
4437     \@tempp
4438 }%

```

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

```

4439     \let\@tempp\@tempm
4440     \edef\@tempm{millies\@tempp}%
4441     \fi
4442     \next
4443 }%
4444 \let\@tempa\@empty
4445 \let\next\@templ
4446 \@templ

```

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

```

4447 \def\@tempb##1{\def\@tempa{\def#2{##1}}}%
4448 \expandafter\@tempb\expandafter{\@tempa}%
4449 \expandafter
4450 }\@tempa
4451 }

```

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.

```

4452 \def\fc@call@opt@arg@second#1#2{%
4453   \def\@tempb{%
4454     \ifx[\@tempa
4455       \def\@tempc[####1]{%
4456         {#1{####1}{#2}}%
4457       }%
4458     \else
4459       \def\@tempc{{#1}{#2}}%
4460     \fi
4461     \@tempc
4462   }%
4463   \futurelet\@tempa
4464   \@tempb
4465 }

4466 \def\fc@call@opt@arg@first#1{%
4467   \def\@tempb{%
4468     \ifx[\@tempa
4469       \def\@tempc[####1]####2{{#1{####1}{####2}}}%
4470     \else
4471       \def\@tempc####1{{#1}{####1}}%
4472     \fi
4473     \@tempc
4474   }%
4475   \futurelet\@tempa
4476   \@tempb
4477 }
4478
4479 \let\fc@call\fc@call@opt@arg@first

```

User API.

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

- #1 local options
- #2 input number

```

4480 \newcommand*{\@latinnumeralstring}[2]{%
4481   \setkeys{fcprefix}{#1}%
4482   \fc@latin@numeral@pexif{#2}\@tempa
4483   \@tempa
4484 }

```

Arguments as follows:

- #1 local options
- #2 input counter

```

4485 \newcommand*{\@latinnumeralstring}[2]{%
4486   \setkeys{fcprefix}{#1}%
4487   \expandafter\let\expandafter
4488     \@tempa\expandafter\csname c@#2\endcsname
4489   \expandafter\fc@latin@numeral@pexif\expandafter{\the\@tempa}\@tempa
4490   \@tempa
4491 }

```



```

4492 \newcommand*{\latinnumeralstring}{%
4493   \fc@call\@latinnumeralstring
4494 }

4495 \newcommand*{\latinnumeralstringnum}{%
4496   \fc@call\@latinnumeralstringnum
4497 }

```

10.4 fmtcount.sty

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

```

4498 \NeedsTeXFormat{LaTeX2e}
4499 \ProvidesPackage{fmtcount}[2020/01/30 v3.06]
4500 \RequirePackage{ifthen}

4501 \RequirePackage{xkeyval}
4502 \RequirePackage{etoolbox}
4503 \RequirePackage{fcprefix}

```

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

```

4504 \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
4505 \providecommand*{\fc@orddef@ult}[1]{\fc@textsuperscript{#1}}

```

```

c@ord@multiling
4506 \providecommand*{\fc@ord@multiling}[1]{%
4507   \ifcsundef{fc@\language@ @alias@of}{%

```

Not a supported language, just use the default setting:

```

4508   \fc@orddef@ult{#1}}{%
4509   \expandafter\let\expandafter\@tempa\csname fc@\language@ @alias@of\endcsname
4510   \ifcsundef{fc@ord@\@tempa}{%

```

Not language specific setting, just use the default setting:

```

4511   \fc@orddef@ult{#1}}{%

```

Language with specific setting, use that setting:

```

4512 \csname fc@ord@\@tempa\endcsname{#1}}}}

```

```

\padzeroes \padzeroes[⟨n⟩]

```

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

```

4513 \newcount\c@padzeroesN
4514 \c@padzeroesN=1\relax
4515 \providecommand*{\padzeroes}[1][17]{\c@padzeroesN=#1}

```

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

Load `fntcount` 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.

```

4516 \newcount\fc@tmpcatcode
4517 \def\fc@languages{}%
4518 \def\fc@mainlang{}%
4519 \newcommand*{\FCloadlang}[1]{%
4520   \@FC@iflangloaded{#1}{}%
4521   {%
4522     \fc@tmpcatcode=\catcode'\@ \relax
4523     \catcode '\@ 11 \relax
4524     \InputIfFileExists{fc-#1.def}%
4525     {%
4526       \ifdefempty{\fc@languages}%
4527       {%
4528         \gdef\fc@languages{#1}%
4529       }%
4530       {%
4531         \gappto\fc@languages{,#1}%
4532       }%
4533       \gdef\fc@mainlang{#1}%
4534     }%
4535   }%
4536   \catcode '\@ \fc@tmpcatcode \relax
4537 }%
4538 }

```

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

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

```

4539 \newcommand{\@FC@iflangloaded}[3]{%
4540   \ifcsundef{ver@fc-#1.def}{#3}{#2}%
4541 }

```

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

```

4542 \newcommand*{\ProvidesFCLanguage}[1]{%
4543   \ProvidesFile{fc-#1.def}%
4544 }

```

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

```

4545 \newif\iffmtcount@language@option
4546 \fmtcount@language@optionfalse

```

`\DeclareLanguageList` 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 L^AT_EX 2_ε.

```

4547 \newcommand*\fc@supported@language@list{%
4548 english,%
4549 UKenglish,%
4550 brazilian,%
4551 british,%
4552 USenglish,%
4553 american,%
4554 spanish,%
4555 portuges,%
4556 portuguese,%
4557 french,%
4558 frenchb,%
4559 francais,%
4560 german,%
4561 germanb,%
4562 ngerman,%
4563 ngermanb,%
4564 italian}

```

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

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

```

4565 \newcommand*\fc@iterate@on@languages[1]{%
4566   \ifx\fc@supported@language@list\@empty

```

That case should never happen !

```

4567     \PackageError{fmtcount}{Macro ‘\protect\fc@iterate@on@languages’ is empty}{You should never
4568       Something is broken within \texttt{fmtcount}, please report the issue on

```

```

4569 \texttt{https://github.com/search?q=fmtcount\&ref=cmdform\&type=Issues}}}%
4570 \else
4571 \let\fc@iterate@on@languages@body#1
4572 \expandafter\@fc@iterate@on@languages\fc@supported@language@list,\@nil,%
4573 \fi
4574 }
4575 \def\@fc@iterate@on@languages#1,{%
4576   {%
4577     \def\@tempa{#1}%
4578     \ifx\@tempa\@nnil
4579       \let\@tempa\@empty
4580     \else
4581       \def\@tempa{%
4582         \fc@iterate@on@languages@body{#1}%
4583         \@fc@iterate@on@languages
4584       }%
4585     \fi
4586     \expandafter
4587   }\@tempa
4588 }%

```

orpolyglossialdf

```
\@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`.

```

4589 \newcommand*\@fc@loadifbabelldf[1]{\ifcsundef{ver@#1.ldf}}{\FCloadlang{#1}}
4590 \newcommand*\@fc@loadifbabelorpolyglossialdf[1]{
4591 \ifpackageloaded{polyglossia}{%
4592 \def\@fc@loadifbabelorpolyglossialdf#1{\IfFileExists{gloss-#1.ldf}{\ifcsundef{#1@loaded}}{\F
4593 \@fc@loadifbabelldf{#1}}%
4594 }%
4595 }\ifpackageloaded{babel}{%
4596 \let\@fc@loadifbabelorpolyglossialdf\@fc@loadifbabelldf
4597 }{}}

```

Load appropriate language definition files:

```
4598 \fc@iterate@on@languages\@fc@loadifbabelorpolyglossialdf
```

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

```

4599 \def\fc@iterate@on@languages@body#1{%
4600 \expandafter\def\csname fc@#1@alias@of\endcsname{#1}}
4601 \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>}`

```
4602 \def\@tempa#1#2{%
4603   \expandafter\def\csname fc@#1@alias@of\endcsname{#2}%
4604 }%
4605 \@tempa{frenchb}{french}
4606 \@tempa{français}{french}
4607 \@tempa{germanb}{german}
4608 \@tempa{ngermanb}{german}
4609 \@tempa{ngerman}{german}
4610 \@tempa{british}{english}
4611 \@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.

```
4612 \def\fc@iterate@on@languages@body#1{%
4613   \define@key{fmtcount}{#1}[]{%
4614     \@FC@iflangloaded{#1}%
4615     {%
4616       \setkeys{fc\csname fc@#1@alias@of\endcsname}{##1}%
4617     }{%
4618       \PackageError{fmtcount}%
4619         {Language ‘#1’ not defined}%
4620         {You need to load \ifxetex polyglossia\else babel\fi\space before loading fmtcount}%
4621     }%
4622   }%
4623   \ifthenelse{\equal{\csname fc@#1@alias@of\endcsname}{#1}}{%
4624     \define@key{fc\csname fc@#1@alias@of\endcsname}{fmtord}{%
4625       \ifthenelse{\equal{##1}{raise}}{or\equal{##1}{level}}{%
4626         \expandafter\let\expandafter\@tempa\csname fc@set@ord@as@##1\endcsname
4627         \expandafter\@tempa\csname fc@ord@#1\endcsname
4628       }{%
4629         \ifthenelse{\equal{##1}{undefine}}{%
4630           \expandafter\let\csname fc@ord@#1\endcsname\undefined
4631         }{%
4632           \PackageError{fmtcount}%
4633             {Invalid value ‘##1’ to fmtord key}%
4634             {Option ‘fmtord’ can only take the values ‘level’, ‘raise’
4635              or ‘undefine’}%
4636         }%
4637       }%
4638     }{%
```

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

```
4639   \expandafter\let\expandafter\@tempa\csname KV@\csname fc@#1@alias@of\endcsname @fmtord\endcsname
4640   \expandafter\let\csname KV@#1@fmtord\endcsname\@tempa
4641 }%
4642 }
4643 \expandafter\@fc@iterate@on@languages\fc@supported@language@list,\@nil,%
```

fmtord Key to determine how to display the ordinal

```

4644 \def\fc@set@ord@as@level#1{%
4645   \def#1##1{##1}%
4646 }
4647 \def\fc@set@ord@as@raise#1{%
4648   \let#1\fc@textsuperscript
4649 }
4650 \define@key{fmtcount}{fmtord}{%
4651   \ifthenelse{\equal{#1}{level}}
4652     \or\equal{#1}{raise}}%
4653   {%
4654     \csname fc@set@ord@as@#1\endcsname\fc@orddef@ult
4655     \def\fmtcount@fmtord{#1}%
4656   }%
4657   {%
4658     \PackageError{fmtcount}%
4659     {Invalid value ‘#1’ to fmtord key}%
4660     {Option ‘fmtord’ can only take the values ‘level’ or ‘raise’}%
4661   }%
4662 }

```

\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.)

```

4663 \newif\iffmtord@abbrv
4664 \fmtord@abbrvtrue
4665 \define@key{fmtcount}{abbrv}[true]{%
4666   \ifthenelse{\equal{#1}{true}\or\equal{#1}{false}}%
4667     {%
4668       \csname fmtord@abbrv#1\endcsname
4669     }%
4670     {%
4671       \PackageError{fmtcount}%
4672       {Invalid value ‘#1’ to fmtord key}%
4673       {Option ‘abbrv’ can only take the values ‘true’ or
4674         ‘false’}%
4675     }%
4676 }

```

prefix

```

4677 \define@key{fmtcount}{prefix}[scale=long]{%
4678   \RequirePackage{fmtprefix}%
4679   \fmtprefixsetoption{#1}%
4680 }

```

countsetoptions Define command to set options.

```

4681 \def\fmtcountsetoptions{%
4682   \def\fmtcount@fmtord{}%

```

```
4683 \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.

```
4684 \InputIfFileExists{fmtcount.cfg}%
4685 {%
4686 \PackageInfo{fmtcount}{Using configuration file fmtcount.cfg}%
4687 }%
4688 {%
4689 }
```

ption@lang@list

```
4690 \newcommand*{\fmtcount@loaded@by@option@lang@list}{}
```

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

```
4691 \newcommand*{\fc@declare@language@option[1]}{%
4692 \DeclareOption{#1}{%
4693 \ifx\fmtcount@loaded@by@option@lang@list\@empty
4694 \def\fmtcount@loaded@by@option@lang@list{#1}%
4695 \else
4696 \edef\fmtcount@loaded@by@option@lang@list{\fmtcount@loaded@by@option@lang@list,#1}%
4697 \fi
4698 }}%
4699 \fc@iterate@on@languages\fc@declare@language@option
```

level

```
4700 \DeclareOption{level}{\def\fmtcount@fmtord{level}%
4701 \def\fc@orddef@ult#1{#1}}
```

raise

```
4702 \DeclareOption{raise}{\def\fmtcount@fmtord{raise}%
4703 \def\fc@orddef@ult#1{\fc@textsuperscript{#1}}}
```

Process package options

```
4704 \ProcessOptions\relax
```

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

```
4705 \ifx\fmtcount@loaded@by@option@lang@list\@empty\else
4706 \def\fc@iterate@on@languages@body#1{%
4707 \FC@iflangloaded{#1}{}%
4708 \fmtcount@language@optiontrue
4709 \FCloadlang{#1}%
4710 }}
4711 \expandafter\fc@iterate@on@languages\fmtcount@loaded@by@option@lang@list,\@nil,%
4712 \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.)

```

4713 \newcount\@DT@modctr
4714 \newcommand*{\@FCmodulo}[2]{%
4715   \@DT@modctr=#1\relax
4716   \divide \@DT@modctr by #2\relax
4717   \multiply \@DT@modctr by #2\relax
4718   \advance #1 by -\@DT@modctr
4719 }

```

The following registers are needed by \@ordinal etc

```

4720 \newcount\@ordinalctr
4721 \newcount\@orgargctr
4722 \newcount\@strctr
4723 \newcount\@tmpstrctr

```

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

```

4724 \newif\if@DT@padzeroes
4725 \newcount\@DT@loopN
4726 \newcount\@DT@X

```

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

```

4727 \newrobustcmd*{\@binary}[1]{%
4728   \@DT@padzeroestrue
4729   \@DT@loopN=17\relax
4730   \@strctr=\@DT@loopN
4731   \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by \@ne}%
4732   \@strctr=65536\relax
4733   \@DT@X=#1\relax
4734   \loop
4735     \@DT@modctr=\@DT@X
4736     \divide \@DT@modctr by \@strctr
4737     \ifthenelse{\boolean{@DT@padzeroes}}
4738       \and \(\@DT@modctr=0\))
4739       \and \(\@DT@loopN>\c@padzeroesN\))}%
4740   {}%
4741   {\the\@DT@modctr}%
4742   \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
4743   \multiply \@DT@modctr by \@strctr
4744   \advance \@DT@X by -\@DT@modctr
4745   \divide \@strctr by \tw@
4746   \advance \@DT@loopN by \m@ne
4747   \ifnum\@strctr>\@ne
4748     \repeat
4749   \the\@DT@X
4750 }
4751

```



```
4752 \let\binarynum=\@binary
```

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

```
4753 \newrobustcmd*{\@octal}[1]{%
4754   \@DT@X=#1\relax
4755   \ifnum\@DT@X>32768
4756     \PackageError{fmtcount}%
4757       {Value of counter too large for \protect\@octal}
4758       {Maximum value 32768}
4759   \else
4760     \@DT@padzeroestrue
4761     \@DT@loopN=6\relax
4762     \@strctr=\@DT@loopN
4763     \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by \@ne}%
4764     \@strctr=32768\relax
4765     \loop
4766       \@DT@modctr=\@DT@X
4767       \divide\@DT@modctr by \@strctr
4768       \ifthenelse{\boolean{\@DT@padzeroes}
4769         \and \(\@DT@modctr=0\)}
4770         \and \(\@DT@loopN>\c@padzeroesN\)}%
4771       {\the\@DT@modctr}%
4772       \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
4773       \multiply\@DT@modctr by \@strctr
4774       \advance\@DT@X by -\@DT@modctr
4775       \divide\@strctr by \@viiipt
4776       \advance\@DT@loopN by \m@ne
4777     \ifnum\@strctr>\@ne
4778     \repeat
4779     \the\@DT@X
4780   \fi
4781 }
4782 \let\octalnum=\@octal
```

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

```
4783 \newcommand*{\@@hexadecimal}[1]{%
4784   \ifcase#1\or1\or2\or3\or4\or5\or
4785   6\or7\or8\or9\or a\or b\or c\or d\or e\or f\fi
4786 }
```

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

```
4787 \newrobustcmd*{\hexadecimalnum}{\@hexadecimalengine\@@hexadecimal}
```

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

```
4788 \newcommand*{\@@Hexadecimal}[1]{%
4789   \ifcase#1\or1\or2\or3\or4\or5\or6\or
4790   7\or8\or9\or A\or B\or C\or D\or E\or F\fi
4791 }
```

\HEXADecimalnum Uppercase hexadecimal

```

4792 \newrobustcmd*{\HEXADecimalnum}{\@hexadecimalengine\@Hexadecimal}
4793 \newcommand*{\@hexadecimalengine}[2]{%
4794   \@DT@padzeroestru
4795   \@DT@loopN=\@vpt
4796   \@strctr=\@DT@loopN
4797   \whiledo{\@strctr<\c@padzeroesN}{0\advance\@strctr by \@ne}%
4798   \@strctr=65536\relax
4799   \@DT@X=#2\relax
4800   \loop
4801     \@DT@modctr=\@DT@X
4802     \divide\@DT@modctr by \@strctr
4803     \ifthenelse{\boolean{\@DT@padzeroes}
4804       \and \(\@DT@modctr=0\)}
4805       \and \(\@DT@loopN>\c@padzeroesN\)}
4806     {\#1\@DT@modctr}%
4807     \ifnum\@DT@modctr=0\else\@DT@padzeroesfalse\fi
4808     \multiply\@DT@modctr by \@strctr
4809     \advance\@DT@X by -\@DT@modctr
4810     \divide\@strctr by 16\relax
4811     \advance\@DT@loopN by \@ne
4812   \ifnum\@strctr>\@ne
4813   \repeat
4814   #1\@DT@X
4815 }
4816 \def\Hexadecimalnum{%
4817   \PackageWarning{fmtcount}{\string\Hexadecimalnum\space is deprecated, use \string\HEXADecimal
4818     instead. The \string\Hexadecimalnum\space control sequence name is confusing as it can misl
4819     that only the 1st letter is upper-cased.}%
4820   \HEXADecimalnum}

```

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

```

4821 \newrobustcmd*{\@aaalph}{\fc@aaalph\@alph}
4822 \newcommand*\fc@aaalph[2]{%
4823   \@DT@loopN=#2\relax
4824   \@DT@X\@DT@loopN
4825   \advance\@DT@loopN by \@ne
4826   \divide\@DT@loopN by 26\relax
4827   \@DT@modctr=\@DT@loopN
4828   \multiply\@DT@modctr by 26\relax
4829   \advance\@DT@X by \@ne
4830   \advance\@DT@X by -\@DT@modctr
4831   \advance\@DT@loopN by \@ne
4832   \advance\@DT@X by \@ne
4833   \edef\@tempa{\#1\@DT@X}%
4834   \loop
4835     \@tempa
4836     \advance\@DT@loopN by \@ne
4837   \ifnum\@DT@loopN>0

```

```

4838 \repeat
4839 }
4840
4841 \let\aaalphnum=\@aaalph

```

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

```

4842 \newrobustcmd*{\@AAAAlph}{\fc@aaalph\@Alph}%
4843
4844 \let\AAAAlphnum=\@AAAAlph

```

\abalphnum Lowercase alphabetical representation

```

4845 \newrobustcmd*{\@abalph}{\fc@abalph\@alph}%
4846 \newcommand*\fc@abalph[2]{%
4847   \@DT@X=#2\relax
4848   \ifnum\@DT@X>17576\relax
4849     \ifx#1\@alph\def\@tempa{\@abalph}%
4850     \else\def\@tempa{\@ABAlph}\fi
4851     \PackageError{fmtcount}%
4852     {Value of counter too large for \expandafter\protect\@tempa}%
4853     {Maximum value 17576}%
4854   \else
4855     \@DT@padzeroestrue
4856     \@strctr=17576\relax
4857     \advance\@DT@X by \@ne
4858     \loop
4859       \@DT@modctr=\@DT@X
4860       \divide\@DT@modctr by \@strctr
4861       \ifthenelse{\boolean{\@DT@padzeroes}}
4862         \and \!(\@DT@modctr=1\)}%
4863       {\#1\@DT@modctr}%
4864       \ifnum\@DT@modctr=\@ne\else\@DT@padzeroesfalse\fi
4865       \multiply\@DT@modctr by \@strctr
4866       \advance\@DT@X by -\@DT@modctr
4867       \divide\@strctr by 26\relax
4868     \ifnum\@strctr>\@ne
4869       \repeat
4870     \advance\@DT@X by \@ne
4871     #1\@DT@X
4872   \fi
4873 }
4874
4875 \let\abalphnum=\@abalph

```

\ABAlphnum Uppercase alphabetical representation

```

4876 \newrobustcmd*{\@ABAlph}{\fc@abalph\@Alph}%
4877 \let\ABAlphnum=\@ABAlph

```

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

```

4878 \def\@fmtc@count#1#2\relax{%
4879   \if\relax#1%
4880   \else
4881     \advance\@strctr by 1\relax
4882     \@fmtc@count#2\relax
4883   \fi
4884 }

```

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

```

4885 \newrobustcmd*{\@decimal}[1]{%
4886   \@strctr=0\relax
4887   \expandafter\@fmtc@count\number#1\relax
4888   \@DT@loopN=\c@padzeroesN
4889   \advance\@DT@loopN by -\@strctr
4890   \ifnum\@DT@loopN>0\relax
4891     \@strctr=0\relax
4892     \whiledo{\@strctr < \@DT@loopN}{0\advance\@strctr by 1\relax}%
4893   \fi
4894   \number#1\relax
4895 }
4896
4897 \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.

```

4898 \newcommand{\FCordinal}[1]{%
4899   \ordinalnum{%
4900     \the\value{#1}}%
4901 }

```

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

```

4902 \ifcsundef{ordinal}
4903 {\let\ordinal\FCordinal}%
4904 {%
4905   \PackageWarning{fmtcount}%
4906   {\protect\ordinal \space already defined use

```

```

4907 \protect\FCordinal \space instead.}
4908 }

```

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

```

4909 \newrobustcmd*{\ordinalnum}[1]{%
4910 \new@ifnextchar[%
4911 {\@ordinalnum{#1}}}%
4912 {\@ordinalnum{#1}[m]}%
4913 }

```

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

```

4914 \def\@ordinalnum#1[#2]{%
4915 {%
4916 \ifthenelse{\equal{#2}{f}}{%
4917 {%
4918 \protect\@ordinalF{#1}{\@fc@ordstr}%
4919 }%
4920 {%
4921 \ifthenelse{\equal{#2}{n}}{%
4922 {%
4923 \protect\@ordinalN{#1}{\@fc@ordstr}%
4924 }%
4925 {%
4926 \ifthenelse{\equal{#2}{m}}{%
4927 {}%
4928 {%
4929 \PackageError{fmtcount}%
4930 {Invalid gender option ‘#2’}%
4931 {Available options are m, f or n}%
4932 }%
4933 \protect\@ordinalM{#1}{\@fc@ordstr}%
4934 }%
4935 }%
4936 \@fc@ordstr
4937 }%
4938 }

```

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

```

4939 \newcommand*{\storeordinal}[2]{%
4940 {%
4941 \toks0{\storeordinalnum{#1}}%
4942 \expandafter
4943 }\the\toks0\expandafter{%
4944 \the\value{#2}}%
4945 }

```

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.)

```
4946 \newrobustcmd*{\storeordinalnum}[2]{%
4947   \@ifnextchar[%
4948     {\@storeordinalnum{#1}{#2}}%
4949     {\@storeordinalnum{#1}{#2}[m]}}%
4950 }
```

`\storeordinalnum` Store ordinal according to gender:

```
4951 \def\@storeordinalnum#1#2[#3]{%
4952   \ifthenelse{\equal{#3}{f}}{%
4953     {%
4954       \protect\@ordinalF{#2}{\@fc@ord}%
4955     }%
4956     {%
4957       \ifthenelse{\equal{#3}{n}}{%
4958         {%
4959           \protect\@ordinalN{#2}{\@fc@ord}%
4960         }%
4961         {%
4962           \ifthenelse{\equal{#3}{m}}{%
4963             {}%
4964             {%
4965               \PackageError{fmtcount}%
4966                 {Invalid gender option ‘#3’}%
4967                 {Available options are m or f}%
4968             }%
4969           \protect\@ordinalM{#2}{\@fc@ord}%
4970         }%
4971       }%
4972     \expandafter\let\csname @fcs@#1\endcsname\@fc@ord
4973 }
```

`\FMCuse` Get stored information:

```
4974 \newcommand*{\FMCuse}[1]{\csname @fcs@#1\endcsname}
```

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

```
4975 \newcommand*{\ordinalstring}[1]{%
4976   \ordinalstringnum{\expandafter\expandafter\expandafter
4977     \the\value{#1}}%
4978 }
```

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

```
4979 \newrobustcmd*{\ordinalstringnum}[1]{%
4980   \new@ifnextchar[%
4981     {\@ordinal@string{#1}}%
4982     {\@ordinal@string{#1}[m]}}%
4983 }
```

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

```

4984 \def\@ordinal@string#1[#2]{%
4985   {%
4986     \ifthenelse{\equal{#2}{f}}{%
4987       {%
4988         \protect\@ordinalstringF{#1}{\@fc@ordstr}%
4989       }%
4990     }%
4991     \ifthenelse{\equal{#2}{n}}{%
4992       {%
4993         \protect\@ordinalstringN{#1}{\@fc@ordstr}%
4994       }%
4995     }%
4996     \ifthenelse{\equal{#2}{m}}{%
4997       }%
4998     {%
4999       \PackageError{fmtcount}%
5000         {Invalid gender option ‘#2’ to \protect\ordinalstring}%
5001         {Available options are m, f or n}%
5002     }%
5003     \protect\@ordinalstringM{#1}{\@fc@ordstr}%
5004   }%
5005 }%
5006 \@fc@ordstr
5007 }%
5008 }

```

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

```

5009 \newcommand*{\storeordinalstring}[2]{%
5010   {%
5011     \toks0{\storeordinalstringnum{#1}}%
5012     \expandafter
5013     }\the\toks0\expandafter{\the\value{#2}}%
5014   }

```

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

```

5015 \newrobustcmd*{\storeordinalstringnum}[2]{%
5016   \@ifnextchar[%
5017     {\@store@ordinal@string{#1}{#2}}%
5018     {\@store@ordinal@string{#1}{#2}[m]}%
5019 }

```

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

```

5020 \def\@store@ordinal@string#1#2[#3]{%
5021   \ifthenelse{\equal{#3}{f}}%
5022   {%

```

```

5023   \protect\@ordinalstringF{#2}{\@fc@ordstr}%
5024 }%
5025 {%
5026   \ifthenelse{\equal{#3}{n}}%
5027   {%
5028     \protect\@ordinalstringN{#2}{\@fc@ordstr}%
5029   }%
5030   {%
5031     \ifthenelse{\equal{#3}{m}}%
5032     {}%
5033     {%
5034       \PackageError{fmtcount}%
5035       {Invalid gender option ‘#3’ to \protect\ordinalstring}%
5036       {Available options are m, f or n}%
5037     }%
5038     \protect\@ordinalstringM{#2}{\@fc@ordstr}%
5039   }%
5040 }%
5041 \expandafter\let\csname @fcs@#1\endcsname\@fc@ordstr
5042 }

```

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

```

5043 \newcommand*{\Ordinalstring}[1]{%
5044   \Ordinalstringnum{\expandafter\expandafter\expandafter\the\value{#1}}%
5045 }

```

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

```

5046 \newrobustcmd*{\Ordinalstringnum}[1]{%
5047   \new@ifnextchar[%
5048   {\@Ordinal@string{#1}}%
5049   {\@Ordinal@string{#1}[m]}%
5050 }

```

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

```

5051 \def\@Ordinal@string#1[#2]{%
5052   {%
5053     \ifthenelse{\equal{#2}{f}}%
5054     {%
5055       \protect\@OrdinalstringF{#1}{\@fc@ordstr}%
5056     }%
5057     {%
5058       \ifthenelse{\equal{#2}{n}}%
5059       {%
5060         \protect\@OrdinalstringN{#1}{\@fc@ordstr}%
5061       }%
5062       {%
5063         \ifthenelse{\equal{#2}{m}}%
5064         {}%

```



```

5065      {%
5066      \PackageError{fmtcount}%
5067      {Invalid gender option ‘#2’}%
5068      {Available options are m, f or n}%
5069      }%
5070      \protect\@OrdinalstringM{#1}{\@fc@ordstr}%
5071    }%
5072  }%
5073  \@fc@ordstr
5074 }%
5075 }

```

reOrdinalstring 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.

```

5076 \newcommand*{\storeOrdinalstring}[2]{%
5077   {%
5078   \toks0{\storeOrdinalstringnum{#1}}%
5079   \expandafter
5080   }\the\toks0\expandafter{\the\value{#2}}%
5081 }

```

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

```

5082 \newrobustcmd*{\storeOrdinalstringnum}[2]{%
5083   \@ifnextchar[%
5084   {\@store@Ordinal@string{#1}{#2}}%
5085   {\@store@Ordinal@string{#1}{#2}[m]}%
5086 }

```

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

```

5087 \def\@store@Ordinal@string#1#2[#3]{%
5088   \ifthenelse{\equal{#3}{f}}{%
5089     {%
5090       \protect\@OrdinalstringF{#2}{\@fc@ordstr}%
5091     }%
5092     {%
5093       \ifthenelse{\equal{#3}{n}}{%
5094         {%
5095           \protect\@OrdinalstringN{#2}{\@fc@ordstr}%
5096         }%
5097         {%
5098           \ifthenelse{\equal{#3}{m}}{%
5099             {}%
5100             {%
5101               \PackageError{fmtcount}%
5102               {Invalid gender option ‘#3’}%
5103               {Available options are m or f}%
5104             }%
5105             \protect\@OrdinalstringM{#2}{\@fc@ordstr}%

```

```

5106     }%
5107 }%
5108 \expandafter\let\csname @fcs@#1\endcsname\@fc@ordstr
5109 }

```

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

```

5110 \newcommand*{\storeORDINALstring}[2]{%
5111   {%
5112     \toks0{\storeORDINALstringnum{#1}}%
5113     \expandafter
5114   }\the\toks0\expandafter{\the\value{#2}}%
5115 }

```

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

```

5116 \newrobustcmd*{\storeORDINALstringnum}[2]{%
5117   \@ifnextchar[%
5118   {\@store@ORDINAL@string{#1}{#2}}%
5119   {\@store@ORDINAL@string{#1}{#2}[m]}%
5120 }

```

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

```

5121 \def\@store@ORDINAL@string#1#2[#3]{%
5122   \ifthenelse{\equal{#3}{f}}{%
5123     {%
5124       \protect\@ordinalstringF{#2}{\@fc@ordstr}%
5125     }%
5126     {%
5127       \ifthenelse{\equal{#3}{n}}{%
5128         {%
5129           \protect\@ordinalstringN{#2}{\@fc@ordstr}%
5130         }%
5131         {%
5132           \ifthenelse{\equal{#3}{m}}{%
5133             {}%
5134             {%
5135               \PackageError{fmtcount}%
5136               {Invalid gender option ‘#3’}%
5137               {Available options are m or f}%
5138             }%
5139           \protect\@ordinalstringM{#2}{\@fc@ordstr}%
5140         }%
5141       }%
5142     }%
5143     \expandafter\protected@edef\csname @fcs@#1\endcsname{%
5144       \noexpand\MakeUppercase{\@fc@ordstr}%
5145     }%
5146   }%

```

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

```
5146 \newcommand*{\ORDINALstring}[1]{%
5147   \ORDINALstringnum{\expandafter\expandafter\expandafter
5148     \the\value{#1}}%
5149 }%
5150 }
```

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

```
5151 \newrobustcmd*{\ORDINALstringnum}[1]{%
5152   \new@ifnextchar[%
5153     {\@ORDINAL@string{#1}}%
5154     {\@ORDINAL@string{#1}[m]}}%
5155 }
```

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

```
5156 \def\@ORDINAL@string#1[#2]{%
5157   {%
5158     \ifthenelse{\equal{#2}{f}}%
5159     {%
5160       \protect\@ordinalstringF{#1}{\@fc@ordstr}%
5161     }%
5162     {%
5163       \ifthenelse{\equal{#2}{n}}%
5164       {%
5165         \protect\@ordinalstringN{#1}{\@fc@ordstr}%
5166       }%
5167       {%
5168         \ifthenelse{\equal{#2}{m}}%
5169         {%
5170           {%
5171             \PackageError{fmtcount}%
5172               {Invalid gender option ‘#2’}%
5173               {Available options are m, f or n}%
5174           }%
5175           \protect\@ordinalstringM{#1}{\@fc@ordstr}%
5176         }%
5177       }%
5178     }%
5179   }%
5180 }
```

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

```
5181 \newcommand*{\storenumberstring}[2]{%
5182   \expandafter\protect\expandafter\storenumberstringnum{#1}{%
5183     \expandafter\the\value{#2}}%
5184 }
```

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

```

5185 \newcommand{\storenumberstringnum}[2]{%
5186   \ifnextchar[%
5187     {\@store@number@string{#1}{#2}}%
5188     {\@store@number@string{#1}{#2}[m]}%
5189 }

```

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

```

5190 \def\@store@number@string#1#2[#3]{%
5191   \ifthenelse{\equal{#3}{f}}{%
5192     {%
5193       \protect\@numberstringF{#2}{\@fc@numstr}%
5194     }%
5195     {%
5196       \ifthenelse{\equal{#3}{n}}{%
5197         {%
5198           \protect\@numberstringN{#2}{\@fc@numstr}%
5199         }%
5200         {%
5201           \ifthenelse{\equal{#3}{m}}{%
5202             {}%
5203             {%
5204               \PackageError{fmtcount}
5205                 {Invalid gender option ‘#3’}%
5206                 {Available options are m, f or n}%
5207             }%
5208           \protect\@numberstringM{#2}{\@fc@numstr}%
5209         }%
5210       }%
5211       \expandafter\let\csname @fcs@#1\endcsname\@fc@numstr
5212 }

```

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

```

5213 \newcommand*{\numberstring}[1]{%
5214   \numberstringnum{\expandafter\expandafter\expandafter
5215     \the\value{#1}}%
5216 }

```

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

```

5217 \newrobustcmd*{\numberstringnum}[1]{%
5218   \new@ifnextchar[%
5219     {\@number@string{#1}}%
5220     {\@number@string{#1}[m]}%
5221 }

```

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

```

5222 \def\@number@string#1[#2]{%
5223   {%
5224     \ifthenelse{\equal{#2}{f}}{%
5225       {%

```

```

5226     \protect\@numberstringF{#1}{\@fc@numstr}%
5227 }%
5228 {%
5229     \ifthenelse{\equal{#2}{n}}%
5230     {%
5231         \protect\@numberstringN{#1}{\@fc@numstr}%
5232     }%
5233     {%
5234         \ifthenelse{\equal{#2}{m}}%
5235         {}%
5236         {%
5237             \PackageError{fmtcount}%
5238             {Invalid gender option ‘#2’}%
5239             {Available options are m, f or n}%
5240         }%
5241         \protect\@numberstringM{#1}{\@fc@numstr}%
5242     }%
5243 }%
5244 \@fc@numstr
5245 }%
5246 }

```

oreNumberstring Store textual representation of number. First argument is identifying name, second argument is a counter.

```

5247 \newcommand*\storeNumberstring[2]{%
5248   {%
5249     \toks0{\storeNumberstringnum{#1}}%
5250     \expandafter
5251   }\the\toks0\expandafter{\the\value{#2}}%
5252 }

```

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

```

5253 \newcommand{\storeNumberstringnum}[2]{%
5254   \@ifnextchar[%
5255   {\@store@Number@string{#1}{#2}}%
5256   {\@store@Number@string{#1}{#2}[m]}%
5257 }

```

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

```

5258 \def\@store@Number@string#1#2[#3]{%
5259   \ifthenelse{\equal{#3}{f}}%
5260   {%
5261     \protect\@NumberstringF{#2}{\@fc@numstr}%
5262   }%
5263   {%
5264     \ifthenelse{\equal{#3}{n}}%
5265     {%
5266       \protect\@NumberstringN{#2}{\@fc@numstr}%
5267     }%

```

```

5268   {%
5269     \ifthenelse{\equal{#3}{m}}%
5270     {}%
5271     {%
5272       \PackageError{fmtcount}%
5273       {Invalid gender option ‘#3’}%
5274       {Available options are m, f or n}%
5275     }%
5276     \protect\@NumberstringM{#2}{\@fc@numstr}%
5277   }%
5278 }%
5279 \expandafter\let\csname @fcs@#1\endcsname\@fc@numstr
5280 }

```

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

```

5281 \newcommand*{\Numberstring}[1]{%
5282   \Numberstringnum{\expandafter\expandafter\expandafter
5283     \the\value{#1}}%
5284 }

```

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

```

5285 \newrobustcmd*{\Numberstringnum}[1]{%
5286   \new@ifnextchar[%
5287     {\@Number@string{#1}}%
5288     {\@Number@string{#1}[m]}%
5289 }

```

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

```

5290 \def\@Number@string#1[#2]{%
5291   {%
5292     \ifthenelse{\equal{#2}{f}}%
5293     {%
5294       \protect\@NumberstringF{#1}{\@fc@numstr}%
5295     }%
5296     {%
5297       \ifthenelse{\equal{#2}{n}}%
5298       {%
5299         \protect\@NumberstringN{#1}{\@fc@numstr}%
5300       }%
5301       {%
5302         \ifthenelse{\equal{#2}{m}}%
5303         {}%
5304         {%
5305           \PackageError{fmtcount}%
5306           {Invalid gender option ‘#2’}%
5307           {Available options are m, f or n}%
5308         }%
5309         \protect\@NumberstringM{#1}{\@fc@numstr}%
5310       }%

```

```

5311     }%
5312     \@fc@numstr
5313 }%
5314 }

```

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

```

5315 \newcommand{\storeNUMBERstring}[2]{%
5316   {%
5317     \toks0{\storeNUMBERstringnum{#1}}%
5318     \expandafter
5319     }\the\toks0\expandafter{\the\value{#2}}%
5320 }

```

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

```

5321 \newcommand{\storeNUMBERstringnum}[2]{%
5322   \@ifnextchar[%
5323   {\@store@NUMBER@string{#1}{#2}}%
5324   {\@store@NUMBER@string{#1}{#2}[m]}%
5325 }

```

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

```

5326 \def\@store@NUMBER@string#1#2[#3]{%
5327   \ifthenelse{\equal{#3}{f}}%
5328   {%
5329     \protect\@numberstringF{#2}{\@fc@numstr}%
5330   }%
5331   {%
5332     \ifthenelse{\equal{#3}{n}}%
5333     {%
5334       \protect\@numberstringN{#2}{\@fc@numstr}%
5335     }%
5336     {%
5337       \ifthenelse{\equal{#3}{m}}%
5338       {}%
5339       {%
5340         \PackageError{fmtcount}%
5341         {Invalid gender option ‘#3’}%
5342         {Available options are m or f}%
5343       }%
5344       \protect\@numberstringM{#2}{\@fc@numstr}%
5345     }%
5346   }%
5347   \expandafter\edef\csname @fcs@#1\endcsname{%
5348     \noexpand\MakeUppercase{\@fc@numstr}%
5349   }%
5350 }

```

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

```

5351 \newcommand*{\NUMBERstring}[1]{%
5352   \NUMBERstringnum{\expandafter\expandafter\expandafter
5353     \the\value{#1}}%
5354 }

```

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

```

5355 \newrobustcmd*{\NUMBERstringnum}[1]{%
5356   \new@ifnextchar[%
5357     {\@NUMBER@string{#1}}%
5358     {\@NUMBER@string{#1}[m]}%
5359 }

```

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

```

5360 \def\@NUMBER@string#1[#2]{%
5361   {%
5362     \ifthenelse{\equal{#2}{f}}%
5363     {%
5364       \protect\@numberstringF{#1}{\@fc@numstr}%
5365     }%
5366     {%
5367       \ifthenelse{\equal{#2}{n}}%
5368       {%
5369         \protect\@numberstringN{#1}{\@fc@numstr}%
5370       }%
5371       {%
5372         \ifthenelse{\equal{#2}{m}}%
5373         {}%
5374         {%
5375           \PackageError{fmtcount}%
5376             {Invalid gender option ‘#2’}%
5377             {Available options are m, f or n}%
5378         }%
5379         \protect\@numberstringM{#1}{\@fc@numstr}%
5380       }%
5381     }%
5382     \MakeUppercase{\@fc@numstr}%
5383   }%
5384 }

```

\binary Number representations in other bases. Binary:

```

5385 \providecommand*{\binary}[1]{%
5386   \@binary{\expandafter\expandafter\expandafter
5387     \the\value{#1}}%
5388 }

```

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

```

5389 \providecommand*{\aaalph}[1]{%
5390   \@aaalph{\expandafter\expandafter\expandafter
5391     \the\value{#1}}%

```


5392 }

`\AAAAlph` As before, but upper case.

```
5393 \providecommand*\AAAAlph}[1]{%
5394   \@AAAAlph{\expandafter\expandafter\expandafter
5395     \the\value{#1}}%
5396 }
```

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

```
5397 \providecommand*\abalph}[1]{%
5398   \@abalph{\expandafter\expandafter\expandafter
5399     \the\value{#1}}%
5400 }
```

`\ABAlph` As above, but upper case.

```
5401 \providecommand*\ABAlph}[1]{%
5402   \@ABAlph{\expandafter\expandafter\expandafter
5403     \the\value{#1}}%
5404 }
```

`\hexadecimal` Hexadecimal:

```
5405 \providecommand*\hexadecimal}[1]{%
5406   \hexadecimalnum{\expandafter\expandafter\expandafter
5407     \the\value{#1}}%
5408 }
```

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

```
5409 \providecommand*\HEXADecimal}[1]{%
5410   \HEXADecimalnum{\expandafter\expandafter\expandafter
5411     \the\value{#1}}%
5412 }
5413 \newrobustcmd*\FC@Hexadecimal@warning{%
5414   \PackageWarning{fmtcount}{\string\Hexadecimal\space is deprecated, use \string\HEXADecimal\space
5415     instead. The \string\Hexadecimal\space control sequence name is confusing as it can mislead
5416     that only the 1st letter is upper-cased.}%
5417 }
5418 \def\Hexadecimal{%
5419   \FC@Hexadecimal@warning
5420   \HEXADecimal}
```

`\octal` Octal:

```
5421 \providecommand*\octal}[1]{%
5422   \@octal{\expandafter\expandafter\expandafter
5423     \the\value{#1}}%
5424 }
```

`\decimal` Decimal:

```
5425 \providecommand*\decimal}[1]{%
```

```

5426 \@decimal{\expandafter\expandafter\expandafter
5427   \the\value{#1}}%
5428 }

```

10.4.1 Multilingual 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.

```

5429 \newif\iffc@languagemode@detected
5430 \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.

```

5431 \def\@setdef@ultfmtcount{%
5432   \fc@languagemode@detectedtrue
5433   \ifcsundef{@ordinalMenglish}{\FCloadlang{english}}{}%
5434   \def\@ordinalstringM{\@ordinalstringMenglish}%
5435   \let\@ordinalstringF=\@ordinalstringMenglish
5436   \let\@ordinalstringN=\@ordinalstringMenglish
5437   \def\@OrdinalstringM{\@OrdinalstringMenglish}%
5438   \let\@OrdinalstringF=\@OrdinalstringMenglish
5439   \let\@OrdinalstringN=\@OrdinalstringMenglish
5440   \def\@numberstringM{\@numberstringMenglish}%
5441   \let\@numberstringF=\@numberstringMenglish
5442   \let\@numberstringN=\@numberstringMenglish
5443   \def\@NumberstringM{\@NumberstringMenglish}%
5444   \let\@NumberstringF=\@NumberstringMenglish
5445   \let\@NumberstringN=\@NumberstringMenglish
5446   \def\@ordinalM{\@ordinalMenglish}%
5447   \let\@ordinalF=\@ordinalM
5448   \let\@ordinalN=\@ordinalM
5449   \let\fmtord\fc@orddef@ult
5450 }

```

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

```

5451 \newcommand*{\fc@multiling}[2]{%
5452   \ifcsundef{@#1#2\languagename}%
5453   {% try loading it
5454     \FCloadlang{\languagename}%
5455   }%
5456   {%
5457   }%
5458   \ifcsundef{@#1#2\languagename}%
5459   {%
5460     \PackageWarning{fmtcount}%
5461     {No support for \expandafter\protect\csname #1\endcsname\space for
5462      language '\languagename'}%

```

```

5463 \ifthenelse{\equal{\language\name}{\fc@mainlang}}{%
5464 {%
5465 \FCloadlang{english}%
5466 }%
5467 {%
5468 }%
5469 \ifcsdef{@#1#2\fc@mainlang}%
5470 {%
5471 \csuse{@#1#2\fc@mainlang}%
5472 }%
5473 {%
5474 \PackageWarningNoLine{fmtcount}%
5475 {No languages loaded at all! Loading english definitions}%
5476 \FCloadlang{english}%
5477 \def\fc@mainlang{english}%
5478 \csuse{@#1#2english}%
5479 }%
5480 }%
5481 {%
5482 \csuse{@#1#2\language\name}%
5483 }%
5484 }

```

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

```

5485 \def\@set@multiling@fmtcount{%
5486 \fc@language@detectedtrue
  The masculine version of \numberstring:
5487 \def\@numberstringM{%
5488 \fc@multiling{numberstring}{M}%
5489 }%
  The feminine version of \numberstring:
5490 \def\@numberstringF{%
5491 \fc@multiling{numberstring}{F}%
5492 }%
  The neuter version of \numberstring:
5493 \def\@numberstringN{%
5494 \fc@multiling{numberstring}{N}%
5495 }%
  The masculine version of \Numberstring:
5496 \def\@NumberstringM{%
5497 \fc@multiling{Numberstring}{M}%
5498 }%
  The feminine version of \Numberstring:
5499 \def\@NumberstringF{%
5500 \fc@multiling{Numberstring}{F}%
5501 }%

```

The neuter version of \Numberstring:

```
5502 \def\@NumberstringN{%  
5503   \fc@multiling{Numberstring}{N}%  
5504 }%
```

The masculine version of \ordinal:

```
5505 \def\@ordinalM{%  
5506   \fc@multiling{ordinal}{M}%  
5507 }%
```

The feminine version of \ordinal:

```
5508 \def\@ordinalF{%  
5509   \fc@multiling{ordinal}{F}%  
5510 }%
```

The neuter version of \ordinal:

```
5511 \def\@ordinalN{%  
5512   \fc@multiling{ordinal}{N}%  
5513 }%
```

The masculine version of \ordinalstring:

```
5514 \def\@ordinalstringM{%  
5515   \fc@multiling{ordinalstring}{M}%  
5516 }%
```

The feminine version of \ordinalstring:

```
5517 \def\@ordinalstringF{%  
5518   \fc@multiling{ordinalstring}{F}%  
5519 }%
```

The neuter version of \ordinalstring:

```
5520 \def\@ordinalstringN{%  
5521   \fc@multiling{ordinalstring}{N}%  
5522 }%
```

The masculine version of \Ordinalstring:

```
5523 \def\@OrdinalstringM{%  
5524   \fc@multiling{Ordinalstring}{M}%  
5525 }%
```

The feminine version of \Ordinalstring:

```
5526 \def\@OrdinalstringF{%  
5527   \fc@multiling{Ordinalstring}{F}%  
5528 }%
```

The neuter version of \Ordinalstring:

```
5529 \def\@OrdinalstringN{%  
5530   \fc@multiling{Ordinalstring}{N}%  
5531 }%
```

Make \fmtord language dependent:

```
5532 \let\fmtord\fc@ord@multiling  
5533 }
```

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.

```
5534 \def\fc@check@for@multiling#1:#2\@nil{%
5535   \@ifpackageloaded{#1}{%
5536     #2\@set@multiling@fmtcount
5537   }{}%
5538 }
```

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.

```
5539 \def\fc@loop@on@multiling@pkg#1,{%
5540   \def\@tempb{#1}%
5541   \ifx\@tempb\@nnil
```

We have reached the end of the loop, so stop here.

```
5542     \let\fc@loop@on@multiling@pkg\@empty
5543   \else
```

Make the \@ifpackageloaded test and break the loop if it was positive.

```
5544     \fc@check@for@multiling#1\@nil
5545     \iffc@language@mode@detected
5546     \def\fc@loop@on@multiling@pkg##1\@nil,{}%
5547   \fi
5548 \fi
5549 \fc@loop@on@multiling@pkg
5550 }
```

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.:

```
5551 \AtBeginDocument{%
5552   \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.

```
5553   \unless\iffc@language@mode@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 \language is defined, and activate the multilingual mode only then, and otherwise fall back to default legacy mode.

```
5554     \ifcsundef{language}%
5555     {%
5556       \PackageWarning{fmtcount}{%
5557         ‘\protect\language’ is undefined, you should use a language package such as bab
5558         when loading a language via package option. Reverting to default language.
5559       }%
5560       \setdef@ultfmtcount
5561     }{%
```

```

5562      \@set@multitling@fmtcount
5563

```

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.

```

5564      \@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.

```

5565      \PackageWarning{fmtcount}{%
5566          Setting ‘\protect\language’ to ‘\fc@mainlang’.\MessageBreak
5567          Reason is that ‘\protect\language’ was ‘\language’,\MessageBreak
5568          but ‘\language’ was not loaded by fmtcount,\MessageBreak
5569          whereas ‘\fc@mainlang’ was the last language loaded by fmtcount ;
5570      }%
5571      \let\language\fc@mainlang
5572  }%
5573 }%
5574 \else
5575     \@setdef@ultfmtcount
5576 \fi\fi
5577 }

```

```

5578 \AtBeginDocument{%
5579     \ifcsundef{FBsupR}{\let\fc@textsuperscript\textsuperscript}{\let\fc@textsuperscript\fcup}%
5580 }

```

Backwards compatibility:

```

5581 \let\@ordinal=\@ordinalM
5582 \let\@ordinalstring=\@ordinalstringM
5583 \let\@Ordinalstring=\@OrdinalstringM
5584 \let\@numberstring=\@numberstringM
5585 \let\@Numberstring=\@NumberstringM

```