# Babel, a multilingual package for use with LaTeX's standard document classes[*]

Johannes Braams
Kersengaarde 33
2723 BP Zoetermeer
The Netherlands
babel@braams.cistron.nl

Printed May 28, 2005

**Abstract**

The standard distribution of LaTeX contains a number of document classes that are meant to be used, but also serve as examples for other users to create their own document classes. These document classes have become very popular among LaTeX users. But it should be kept in mind that they were designed for American tastes and typography. At one time they contained a number of hard-wired texts. This report describes babel, a package that makes use of the new capabilities of TeX version 3 to provide an environment in which documents can be typeset in a non-American language, or in more than one language.

## Contents

[*]During the development ideas from Nico Poppelier, Piet van Oostrum and many others have been used. Bernd Raichle has provided many helpful suggestions.

## 1 The user interface

The user interface of this package is quite simple. It consists of a set of commands that switch from one language to another, and a set of commands that deal with shorthands. It is also possible to find out what the current language is.

`\selectlanguage` When a user wants to switch from one language to another he can do so using the macro `\selectlanguage`. This macro takes the language, defined previously by a language definition file, as its argument. It calls several macros that should be defined in the language definition files to activate the special definitions for the language chosen.

`otherlanguage` The environment otherlanguage does basically the same as `\selectlanguage`, except the language change is local to the environment. This environment is required for intermixing left-to-right typesetting with right-to-left typesetting. The language to switch to is specified as an argument to `\begin{otherlanguage}`.

`\foreignlanguage` The command `\foreignlanguage` takes two arguments; the second argument is a phrase to be typeset according to the rules of the language named in its first argument. This command only switches the extra definitions and the hyphenation rules for the language, *not* the names and dates.

`otherlanguage*` In the environment otherlanguage* only the typesetting is done according to the rules of the other language, but the text-strings such as 'figure', 'table', etc. are left as they were set outside this environment.

`hyphenrules` The environment hyphenrules can be used to select *only* the hyphenation rules to be used. This can for instance be used to select 'nohyphenation', provided that in `language.dat` the 'language' nohyphenation is defined by loading `serohyph.tex`.

`\languagename` The control sequence `\languagename` contains the name of the current language.

`\iflanguage` If more than one language is used, it might be necessary to know which language is active at a specific time. This can be checked by a call to `\iflanguage`. This macro takes three arguments. The first argument is the name of a language; the second and third arguments are the actions to take if the result of the test is `true` or `false` respectively.

`\useshorthands` The command `\useshorthands` initiates the definition of user-defined shorthand sequences. It has one argument, the character that starts these personal shorthands.

`\defineshorthand` The command `\defineshorthand` takes two arguments: the first is a one- or two-character shorthand sequence, and the second is the code the shorthand should expand to.

`\aliasshorthand` The command `\aliasshorthand` can be used to let another character perform the same functions as the default shorthand character. If one prefers for example to use the character / over " in typing polish texts, this can be achieved by entering `\aliasshorthand{"}{/}`. *Please note* that the substitute shorthand character must have been declared in the preamble of your document, using a command such as `\useshorthands{/}` in this example.

`\languageshorthands` The command `\languageshorthands` can be used to switch the shorthands on the language level. It takes one argument, the name of a language. Note that for this to work the language should have been specified as an option when loading the babel package.

`\shorthandon` `\shorthandoff` It is sometimes necessary to switch a shorthand character off temporarily, because it must be used in an entirely different way. For this purpose, the user commands `\shorthandoff` and `\shorthandon` are provided. They each take a list of characters as their arguments. The command `\shorthandoff` sets the `\catcode` for each of the characters in its argument to other (12); the command `\shorthandon` sets the `\catcode` to active (13). Both commands only work on 'known' shorthand characters. If a character is not known to be a shorthand character its category code will be left unchanged.

`\languageattribute` This is a user-level command, to be used in the preamble of a document (after `\usepackage[...]{babel}`), that declares which attributes are to be used for a given language. It takes two arguments: the first is the name of the language; the second, a (list of) attribute(s) to used. The command checks whether the language is known in this document and whether the attribute(s) are known for this language.

## 1.1 Languages supported by Babel

In the following table all the languages supported by Babel are listed, together with the names of the options with which you can load babel for each language.

| Language | Option(s) |
|---|---|
| Afrikaans | afrikaans |
| Bahasa | bahasa |
| Basque | basque |
| Breton | breton |
| Bulgarian | bulgarian |
| Catalan | catalan |
| Croatian | croatian |
| Czech | czech |
| Danish | danish |
| Dutch | dutch |
| English | english, USenglish, american, UKenglish, british, canadian, australian, newzealand |
| Esperanto | esperanto |
| Estonian | estonian |
| Finnish | finnish |
| French | french, francais, canadien, acadian |
| Galician | galician |
| German | austrian, german, germanb, ngerman, naustrian |
| Greek | greek, polutonikogreek |
| Hebrew | hebrew |
| Hungarian | magyar, hungarian |
| Icelandic | icelandic |

| Language | Option(s) |
|---|---|
| Interlingua | interlingua |
| Irish Gaelic | irish |
| Italian | italian |
| Latin | latin |
| Lower Sorbian | lowersorbian |
| North Sami | samin |
| Norwegian | norsk, nynorsk |
| Polish | polish |
| Portuguese | portuges, portuguese, brazilian, brazil |
| Romanian | romanian |
| Russian | russian |
| Scottish Gaelic | scottish |
| Spanish | spanish |
| Slovakian | slovak |
| Slovenian | slovene |
| Swedish | swedish |
| Serbian | serbian |
| Turkish | turkish |
| Ukrainian | ukrainian |
| Upper Sorbian | uppersorbian |
| Welsh | welsh |

For some languages babel supports the options activeacute and activegrave; for typestting Russian texts, babel knows about the options LWN and LCY to specify the fontencoding of the cyrillic font used. Currently only LWN is supported.

### 1.2 Workarounds

If you use the document class `book` *and* you use `\ref` inside the argument of `\chapter`, LaTeX will keep complaining about an undefined label. The reason is that the argument of `\ref` is passed through `\uppercase` at some time during processing. To prevent such problems, you could revert to using uppercase labels, or you can use `\lowercase{\ref{foo}}` inside the argument of `\chapter`.

## 2 Changes for LaTeX $2_\varepsilon$

With the advent of LaTeX $2_\varepsilon$ the interface to babel in the preamble of the document has changed. With LaTeX2.09 one used to call up the babel system with a line such as:

```
\documentstyle[dutch,english]{article}
```

which would tell LaTeX that the document would be written in two languages, Dutch and English, and that English would be the first language in use.

The LaTeX $2_\varepsilon$ way of providing the same information is:

```
\documentclass{article}
\usepackage[dutch,english]{babel}
```

or, making dutch and english global options in order to let other packages detect and use them:

```
\documentclass[dutch,english]{article}
\usepackage{babel}
\usepackage{varioref}
```

In this last example, the package `varioref` will also see the options and will be able to use them.

## 3 Changes in Babel version 3.7

In Babel version 3.7 a number of bugs that were found in version 3.6 are fixed. Also a number of changes and additions have occurred:

- Shorthands are expandable again. The disadvantage is that one has to type `'{}a` when the acute accent is used as a shorthand character. The advantage is that a number of other problems (such as the breaking of ligatures, etc.) have vanished.
- Two new commands, `\shorthandon` and `\shorthandoff` have been introduced to enable to temporarily switch off one or more shorthands.
- Support for typesetting Greek has been enhanced. Code from the `kdgreek` package (suggested by the author) was added and `\greeknumeral` has been added.
- Support for typesetting Basque is now available thanks to Juan Aguirregabiria.
- Support for typesetting Serbian with Latin script is now available thanks to Dejan Muhamedagić and Jankovic Slobodan.
- Support for typesetting Hebrew (and potential support for typesetting other right-to-left written languages) is now available thanks to Rama Porrat and Boris Lavva.
- Support for typesetting Bulgarian is now available thanks to Georgi Boshnakov.
- Support for typesetting Latin is now available, thanks to Claudio Beccari and Krzysztof Konrad Żelechowski.
- Support for typesetting North Sami is now available, thanks to Regnor Jernsletten.

- The options canadian, canadien and acadien have been added for Canadian English and French use.
- A language attribute has been added to the `\mark...` commands in order to make sure that a Greek header line comes out right on the last page before a language switch.
- Hyphenation pattern files are now read *inside a group*; therefore any changes a pattern file needs to make to lowercase codes, uppercase codes, and category codes are kept local to that group. If they are needed for the language, these changes will need to be repeated and stored in `\extras...`
- The concept of language attributes is introduced. It is intended to give the user some control over the features a language-definition file provides. Its first use is for the Greek language, where the user can choose the *πολυτονικό* ("Polutoniko" or multi-accented) Greek way of typesetting texts. These attributes will possibly find wider use in future releases.
- The environment hyphenrules is introduced.
- The syntax of the file `language.dat` has been extended to allow (optionally) specifying the font encoding to be used while processing the patterns file.
- The command `\providehyphenmins` should now be used in language definition files in order to be able to keep any settings provided by the pattern file.

## 4 Changes in Babel version 3.6

In Babel version 3.6 a number of bugs that were found in version 3.5 are fixed. Also a number of changes and additions have occurred:

- A new environment otherlanguage* is introduced. it only switches the 'specials', but leaves the 'captions' untouched.
- The shorthands are no longer fully expandable. Some problems could only be solved by peeking at the token following an active character. The advantage is that `'{}a` works as expected for languages that have the `'` active.
- Support for typesetting french texts is much enhanced; the file `francais.ldf` is now replaced by `frenchb.ldf` which is maintained by Daniel Flipo.
- Support for typesetting the russian language is again available. The language definition file was originally developed by Olga Lapko from CyrTUG. The fonts needed to typeset the russian language are now part of the babel distribution. The support is not yet up to the level which is needed according to Olga, but this is a start.

- Support for typesetting greek texts is now also available. What is offered in this release is a first attempt; it will be enhanced later on by Yannis Haralambous.
- in babel 3.6j some hooks have been added for the development of support for Hebrew typesetting.
- Support for typesetting texts in Afrikaans (a variant of Dutch, spoken in South Africa) has been added to `dutch.ldf`.
- Support for typesetting Welsh texts is now available.
- A new command `\aliasshorthand` is introduced. It seems that in Poland various conventions are used to type the necessary Polish letters. It is now possible to use the character `/` as a shorthand character instead of the character `"`, by issuing the command `\aliasshorthand{"}{/}`.
- The shorthand mechanism now deals correctly with characters that are already active.
- Shorthand characters are made active at `\begin{document}`, not earlier. This is to prevent problems with other packages.
- A *preambleonly* command `\substitutefontfamily` has been added to create `.fd` files on the fly when the font families of the Latin text differ from the families used for the Cyrillic or Greek parts of the text.
- Three new commands `\LdfInit`, `\ldf@quit` and `\ldf@finish` are introduced that perform a number of standard tasks.
- In babel 3.6k the language Ukrainian has been added and the support for Russian typesetting has been adapted to the package 'cyrillic' to be released with the December 1998 release of LaTeX $2_\varepsilon$.

## 5 Changes in Babel version 3.5

In Babel version 3.5 a lot of changes have been made when compared with the previous release. Here is a list of the most important ones:

- the selection of the language is delayed until `\begin{document}`, which means you must add appropriate `\selectlanguage` commands if you include `\hyphenation` lists in the preamble of your document.
- babel now has a language environment and a new command `\foreignlanguage`;
- the way active characters are dealt with is completely changed. They are called 'shorthands'; one can have three levels of shorthands: on the user level, the language level, and on 'system level'. A consequence of the new way of handling active characters is that they are now written to auxiliary files 'verbatim';

- A language change now also writes information in the `.aux` file, as the change might also affect typesetting the table of contents. The consequence is that an .aux file generated by a LaTeX format with babel preloaded gives errors when read with a LaTeX format without babel; but I think this probably doesn't occur;
- babel is now compatible with the `inputenc` and `fontenc` packages;
- the language definition files now have a new extension, `ldf`;
- the syntax of the file `language.dat` is extended to be compatible with the `french` package by Bernard Gaulle;
- each language definition file looks for a configuration file which has the same name, but the extension `.cfg`. It can contain any valid LaTeX code.

## 6 The interface between the core of babel and the language definition files

In the core of the babel system, several macros are defined for use in language definition files. Their purpose is to make a new language known.

`\addlanguage` The macro `\addlanguage` is a non-outer version of the macro `\newlanguage`, defined in `plain.tex` version 3.x. For older versions of `plain.tex` and `lplain.tex` a substitute definition is used.

`\adddialect` The macro `\adddialect` can be used when two languages can (or must) use the same hyphenation patterns. This can also be useful for languages for which no patterns are preloaded in the format. In such cases the default behaviour of the babel system is to define this language as a 'dialect' of the language for which the patterns were loaded as `\language0`.

The language definition files must conform to a number of conventions, because these files have to fill in the gaps left by the common code in `babel.def`, i.e., the definitions of the macros that produce texts. Also the language-switching possibility which has been built into the babel system has its implications.

The following assumptions are made:

- Some of the language-specific definitions might be used by plain TeX users, so the files have to be coded so that they can be read by both LaTeX and plain TeX. The current format can be checked by looking at the value of the macro `\fmtname`.
- The common part of the babel system redefines a number of macros and environments (defined previously in the document style) to put in the names of macros that replace the previously hard-wired texts. These macros have to be defined in the language definition files.
- The language definition files define five macros, used to activate and deactivate the language-specific definitions. These macros are \⟨*lang*⟩`hyphenmins`,

`\captions`⟨*lang*⟩, `\date`⟨*lang*⟩, `\extras`⟨*lang*⟩ and `\noextras`⟨*lang*⟩; where ⟨*lang*⟩ is either the name of the language definition file or the name of the LaTeX option that is to be used. These macros and their functions are discussed below.

- When a language definition file is loaded, it can define `\l@`⟨*lang*⟩ to be a dialect of `\language0` when `\l@`⟨*lang*⟩ is undefined.
- The language definition files can be read in the preamble of the document, but also in the middle of document processing. This means that they have to function independently of the current `\catcode` of the `@` sign.

`\providehyphenmins` The macro `\providehyphenmins` should be used in the language definition files to set the `\lefthyphenmin` and `\righthyphenmin`. This macro will check whether these parameters were provided by the hyphenation file before it takes any action.

`\langhyphenmins` The macro `\`⟨*lang*⟩`hyphenmins` is used to store the values of the `\lefthyphenmin` and `\righthyphenmin`.

`\captionslang` The macro `\captions`⟨*lang*⟩ defines the macros that hold the texts to replace the original hard-wired texts.

`\datelang` The macro `\date`⟨*lang*⟩ defines `\today` and

`\extraslang` The macro `\extras`⟨*lang*⟩ contains all the extra definitions needed for a specific language.

`\noextraslang` Because we want to let the user switch between languages, but we do not know what state TeX might be in after the execution of `\extras`⟨*lang*⟩, a macro that brings TeX into a predefined state is needed. It will be no surprise that the name of this macro is `\noextras`⟨*lang*⟩.

`\bbl@declare@ttribute` This is a command to be used in the language definition files for declaring a language attribute. It takes three arguments: the name of the language, the attribute to be defined, and the code to be executed when the attribute is to be used.

`\main@language` To postpone the activation of the definitions needed for a language until the beginning of a document, all language definition files should use `\main@language` instead of `\selectlanguage`. This will just store the name of the language, and the proper language will be activated at the start of the document.

`\ProvidesLanguage` The macro `\ProvidesLanguage` should be used to identify the language definition files. Its syntax is similar to the syntax of the LaTeX command `\ProvidesPackage`.

`\LdfInit` The macro `\LdfInit` performs a couple of standard checks that must be made at the beginning of a language definition file, such as checking the category code of the @-sign, preventing the `.ldf` file from being processed twice, etc.

`\ldf@quit` The macro `\ldf@quit` does work needed if a `.ldf` file was processed earlier. This includes resetting the category code of the @-sign, preparing the language to be activated at `\begin{document}` time, and ending the input stream.

`\ldf@finish` The macro `\ldf@finish` does work needed at the end of each `.ldf` file. This includes resetting the category code of the @-sign, loading a local configuration file, and preparing the language to be activated at `\begin{document}` time.

`\loadlocalcfg` After processing a language definition file, LaTeX can be instructed to load a local configuration file. This file can, for instance, be used to add strings to `\captions`⟨*lang*⟩ to support local document classes. The user will be informed that this configuration file has been loaded. This macro is called by `\ldf@finish`.

`\substitutefontfamily` This command takes three arguments, a font encoding and two font family names. It creates a font description file for the first font in the given encoding. This `.fd` file will instruct LaTeX to use a font from the second family when a font from the first family in the given encoding seems to be needed.

## 6.1 Support for active characters

In quite a number of language definition files, active characters are introduced. To facilitate this, some support macros are provided.

`\initiate@active@char` The internal macro `\initiate@active@char` is used in language definition files to instruct LaTeX to give a character the category code 'active'. When a character has been made active it will remain that way until the end of the document. Its definition may vary.

`\bbl@activate` `\bbl@deactivate` The command `\bbl@activate` is used to change the way an active character expands. `\bbl@activate` 'switches on' the active behaviour of the character. `\bbl@deactivate` lets the active character expand to its former (mostly) non-active self.

`\declare@shorthand` The macro `\declare@shorthand` is used to define the various shorthands. It takes three arguments: the name for the collection of shorthands this definition belongs to; the character (sequence) that makes up the shorthand, i.e. `~` or `"a`; and the code to be executed when the shorthand is encountered.

`\bbl@add@special` `\bbl@remove@special` The TeXbook states: "Plain TeX includes a macro called `\dospecials` that is essentially a set macro, representing the set of all characters that have a special category code." [1, p. 380] It is used to set text 'verbatim'. To make this work if more characters get a special category code, you have to add this character to the macro `\dospecial`. LaTeX adds another macro called `\@sanitize` representing the same character set, but without the curly braces. The macros `\bbl@add@special`⟨*char*⟩ and `\bbl@remove@special`⟨*char*⟩ add and remove the character ⟨*char*⟩ to these two sets.

## 6.2 Support for saving macro definitions

Language definition files may want to *re*define macros that already exist. Therefor a mechanism for saving (and restoring) the original definition of those macros is provided. We provide two macros for this[1].

`\babel@save` To save the current meaning of any control sequence, the macro `\babel@save` is provided. It takes one argument, ⟨*csname*⟩, the control sequence for which the meaning has to be saved.

`\babel@savevariable` A second macro is provided to save the current value of a variable. In this context, anything that is allowed after the `\the` primitive is considered to be a variable. The macro takes one argument, the ⟨*variable*⟩.

[1]This mechanism was introduced by Bernd Raichle.

The effect of the preceding macros is to append a piece of code to the current definition of `\originalTeX`. When `\originalTeX` is expanded, this code restores the previous definition of the control sequence or the previous value of the variable.

## 6.3 Support for extending macros

`\addto` The macro `\addto{`⟨*control sequence*⟩`}{`⟨*TEX code*⟩`}` can be used to extend the definition of a macro. The macro need not be defined. This macro can, for instance, be used in adding instructions to a macro like `\extrasenglish`.

## 6.4 Macros common to a number of languages

`\allowhyphens` In a couple of European languages compound words are used. This means that when TEX has to hyphenate such a compound word, it only does so at the '`-`' that is used in such words. To allow hyphenation in the rest of such a compound word, the macro `\allowhyphens` can be used.

`\set@low@box` For some languages, quotes need to be lowered to the baseline. For this purpose the macro `\set@low@box` is available. It takes one argument and puts that argument in an `\hbox`, at the baseline. The result is available in `\box0` for further processing.

`\save@sf@q` Sometimes it is necessary to preserve the `\spacefactor`. For this purpose the macro `\save@sf@q` is available. It takes one argument, saves the current spacefactor, executes the argument, and restores the spacefactor.

`\bbl@frenchspacing` `\bbl@nonfrenchspacing` The commands `\bbl@frenchspacing` and `\bbl@nonfrenchspacing` can be used to properly switch French spacing on and off.

# 7 Compatibility with `german.sty`

The file `german.sty` has been one of the sources of inspiration for the babel system. Because of this I wanted to include `german.sty` in the babel system. To be able to do that I had to allow for one incompatibility: in the definition of the macro `\selectlanguage` in `german.sty` the argument is used as the ⟨*number*⟩ for an `\ifcase`. So in this case a call to `\selectlanguage` might look like `\selectlanguage{\german}`.

In the definition of the macro `\selectlanguage` in `babel.def` the argument is used as a part of other macronames, so a call to `\selectlanguage` now looks like `\selectlanguage{german}`. Notice the absence of the escape character. As of version 3.1a of babel both syntaxes are allowed.

All other features of the original `german.sty` have been copied into a new file, called `germanb.sty`[2].

Although the babel system was developed to be used with LATEX, some of the features implemented in the language definition files might be needed by plain TEX users. Care has been taken that all files in the system can be processed by plain TEX.

[2]The 'b' is added to the name to distinguish the file from Partls' file.

## 8 Compatibility with `ngerman.sty`

When used with the options ngerman or naustrian, babel will provide all features of the package `ngerman`. There is however one exception: The commands for special hyphenation of double consonants (`"ff` etc.) and ck (`"ck`), which are no longer required with the new German orthography, are undefined. With the `ngerman` package, however, these commands will generate appropriate warning messages only.

## 9 Compatibility with the `french` package

It has been reported to me that the package `french` by Bernard Gaulle (`gaulle@idris.fr`) works together with babel. On the other hand, it seems *not* to work well together with a lot of other packages. Therefore I have decided to no longer load `french.ldf` by default. Instead, when you want to use the package by Bernard Gaulle, you will have to request it specifically, by passing either frenchle or frenchpro as an option to babel.

## 10 Identification

The file `babel.sty`[3] is meant for LaTeX 2$_\varepsilon$, therefor we make sure that the format file used is the right one.

`\ProvidesLanguage` The identification code for each file is something that was introduced in LaTeX 2$_\varepsilon$. When the command `\ProvidesFile` does not exist, a dummy definition is provided temporarily. For use in the language definition file the command `\ProvidesLanguage` is defined by babel.

```
⟨*!package⟩
\ifx\ProvidesFile\@undefined
  \def\ProvidesFile#1[#2 #3 #4]{%
    \wlog{File: #1 #4 #3 <#2>}%
⟨*kernel & patterns⟩
    \toks8{Babel <#3> and hyphenation patterns for }%
⟨/kernel & patterns⟩
    \let\ProvidesFile\@undefined
    }
```

As an alternative for `\ProvidesFile` we define `\ProvidesLanguage` here to be used in the language definition files.

```
⟨*kernel⟩
  \def\ProvidesLanguage#1[#2 #3 #4]{%
    \wlog{Language: #1 #4 #3 <#2>}%
    }
\else
```

[3]The file described in this section is called `babel.dtx`, has version number v3.8g and was last revised on 2005/05/21.

In this case we save the original definition of `\ProvidesFile` in `\bbl@tempa` and restore it after we have stored the version of the file in `\toks8`.

```
⟨*kernel & patterns⟩
  \let\bbl@tempa\ProvidesFile
  \def\ProvidesFile#1[#2 #3 #4]{%
    \toks8{Babel <#3> and hyphenation patterns for }%
    \bbl@tempa#1[#2 #3 #4]%
    \let\ProvidesFile\bbl@tempa}
⟨/kernel & patterns⟩
```

When `\ProvidesFile` is defined we give `\ProvidesLanguage` a similar definition.

```
  \def\ProvidesLanguage#1{%
    \begingroup
      \catcode`\ 10 %
      \@makeother\/%
      \@ifnextchar[%]
        {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
  \def\@provideslanguage#1[#2]{%
    \wlog{Language: #1 #2}%
    \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
    \endgroup}
⟨/kernel⟩
\fi
⟨/!package⟩
```

Identify each file that is produced from this source file.

```
⟨+package⟩\ProvidesPackage{babel}
⟨+core⟩\ProvidesFile{babel.def}
⟨+kernel & patterns⟩\ProvidesFile{hyphen.cfg}
⟨+kernel&!patterns⟩\ProvidesFile{switch.def}
⟨+driver&!user⟩\ProvidesFile{babel.drv}
⟨+driver & user⟩\ProvidesFile{user.drv}
                  [2005/05/21 v3.8g %
⟨+package⟩      The Babel package]
⟨+core⟩          Babel common definitions]
⟨+kernel⟩       Babel language switching mechanism]
⟨+driver⟩]
```

## 11 The Package File

In order to make use of the features of LaTeX $2_\varepsilon$, the babel system contains a package file, `babel.sty`. This file is loaded by the `\usepackage` command and defines all the language options known in the babel system. It also takes care of a number of compatibility issues with other packages.

### 11.1 Language options

```
⟨*package⟩
```

```
\ifx\LdfInit\@undefined\input babel.def\relax\fi
```

For all the languages supported we need to declare an option.

```
\DeclareOption{acadian}{\input{frenchb.ldf}}
\DeclareOption{afrikaans}{\input{dutch.ldf}}
\DeclareOption{american}{\input{english.ldf}}
\DeclareOption{australian}{\input{english.ldf}}
```

Austrian is really a dialect of German.

```
\DeclareOption{austrian}{\input{germanb.ldf}}
\DeclareOption{bahasa}{\input{bahasa.ldf}}
\DeclareOption{basque}{\input{basque.ldf}}
\DeclareOption{brazil}{\input{portuges.ldf}}
\DeclareOption{brazilian}{\input{portuges.ldf}}
\DeclareOption{breton}{\input{breton.ldf}}
\DeclareOption{british}{\input{english.ldf}}
\DeclareOption{bulgarian}{\input{bulgarian.ldf}}
\DeclareOption{canadian}{\input{english.ldf}}
\DeclareOption{canadien}{\input{frenchb.ldf}}
\DeclareOption{catalan}{\input{catalan.ldf}}
\DeclareOption{croatian}{\input{croatian.ldf}}
\DeclareOption{czech}{\input{czech.ldf}}
\DeclareOption{danish}{\input{danish.ldf}}
\DeclareOption{dutch}{\input{dutch.ldf}}
\DeclareOption{english}{\input{english.ldf}}
\DeclareOption{esperanto}{\input{esperanto.ldf}}
\DeclareOption{estonian}{\input{estonian.ldf}}
\DeclareOption{finnish}{\input{finnish.ldf}}
```

The babel support or French used to be stored in `francais.ldf`; therefor the LaTeX2.09 option used to be francais. The hyphenation patterns may be loaded as either 'french' or as 'francais'.

```
\DeclareOption{francais}{\input{frenchb.ldf}}
\DeclareOption{frenchb}{\input{frenchb.ldf}}
```

With LaTeX 2$_\varepsilon$ we can now also use the option french and still call the file `frenchb.ldf`.

```
\DeclareOption{french}{\input{frenchb.ldf}}%
\DeclareOption{galician}{\input{galician.ldf}}
\DeclareOption{german}{\input{germanb.ldf}}
\DeclareOption{germanb}{\input{germanb.ldf}}
\DeclareOption{greek}{\input{greek.ldf}}
\DeclareOption{polutonikogreek}{%
  \input{greek.ldf}%
  \languageattribute{greek}{polutoniko}}
\DeclareOption{hebrew}{%
  \input{rlbabel.def}%
  \input{hebrew.ldf}}
```

hungarian is just a synonym for magyar

```
\DeclareOption{hungarian}{\input{magyar.ldf}}
\DeclareOption{icelandic}{\input{icelandic.ldf}}
\DeclareOption{interlingua}{\input{interlingua.ldf}}
\DeclareOption{irish}{\input{irish.ldf}}
\DeclareOption{italian}{\input{italian.ldf}}
\DeclareOption{latin}{\input{latin.ldf}}
\DeclareOption{lowersorbian}{\input{lsorbian.ldf}}
%^^A\DeclareOption{kannada}{\input{kannada.ldf}}
\DeclareOption{magyar}{\input{magyar.ldf}}
%^^A\DeclareOption{nagari}{\input{nagari.ldf}}
```

'New' German orthography, including Austrian variant:

```
\DeclareOption{naustrian}{\input{ngermanb.ldf}}
\DeclareOption{newzealand}{\input{english.ldf}}
\DeclareOption{ngerman}{\input{ngermanb.ldf}}
\DeclareOption{norsk}{\input{norsk.ldf}}
\DeclareOption{samin}{\input{samin.ldf}}
```

For Norwegian two spelling variants are provided.

```
\DeclareOption{nynorsk}{\input{norsk.ldf}}
\DeclareOption{polish}{\input{polish.ldf}}
\DeclareOption{portuges}{\input{portuges.ldf}}
\DeclareOption{portuguese}{\input{portuges.ldf}}
\DeclareOption{romanian}{\input{romanian.ldf}}
\DeclareOption{russian}{\input{russianb.ldf}}
%^^A\DeclareOption{sanskrit}{\input{sanskrit.ldf}}
\DeclareOption{scottish}{\input{scottish.ldf}}
\DeclareOption{serbian}{\input{serbian.ldf}}
\DeclareOption{slovak}{\input{slovak.ldf}}
\DeclareOption{slovene}{\input{slovene.ldf}}
\DeclareOption{spanish}{\input{spanish.ldf}}
\DeclareOption{swedish}{\input{swedish.ldf}}
%^^A\DeclareOption{tamil}{\input{tamil.ldf}}
\DeclareOption{turkish}{\input{turkish.ldf}}
\DeclareOption{ukrainian}{\input{ukraineb.ldf}}
\DeclareOption{uppersorbian}{\input{usorbian.ldf}}
\DeclareOption{welsh}{\input{welsh.ldf}}
\DeclareOption{UKenglish}{\input{english.ldf}}
\DeclareOption{USenglish}{\input{english.ldf}}
```

For all those languages for which the option name is the same as the name of the language specific file we specify a default option, which tries to load the file specified. If this doesn't succeed an error is signalled.

```
\DeclareOption*{%
  \InputIfFileExists{\CurrentOption.ldf}{}{%
    \PackageError{babel}{%
      Language definition file \CurrentOption.ldf not found}{%
      Maybe you misspelled the language option?}}%
  }
```

Another way to extend the list of 'known' options for babel is to create the file `bblopts.cfg` in which one can add option declarations.

```
\InputIfFileExists{bblopts.cfg}{%
  \typeout{*************************************^^J%
           * Local config file bblopts.cfg used^^J%
           *}%
  }{}
```

Apart from all the language options we also have a few options that influence the behaviour of language definition files.

The following options don't do anything themselves, they are just defined in order to make it possible for language definition files to check if one of them was specified by the user.

```
\DeclareOption{activeacute}{}
\DeclareOption{activegrave}{}
```

The next option tells babel to leave shorthand characters active at the end of processing the package. This is *not* the default as it can cause problems with other packages, but for those who want to use the shorthand characters in the preamble of their documents this can help.

```
\DeclareOption{KeepShorthandsActive}{}
```

The options have to be processed in the order in which the user specified them:

```
\ProcessOptions*
```

In order to catch the case where the user forgot to specify a language we check whether `\bbl@main@language`, has become defined. If not, no language has been loaded and an error message is displayed.

```
\ifx\bbl@main@language\@undefined
  \PackageError{babel}{%
    You haven't specified a language option}{%
    You need to specify a language, either as a global
    option\MessageBreak
    or as an optional argument to the \string\usepackage\space
    command; \MessageBreak
    You shouldn't try to proceed from here, type x to quit.}
```

To prevent undefined command errors when the user insists on continuing we load `babel.def` here. He should expect more errors though.

```
  \input{babel.def}
\fi
```

`\substitutefontfamily` The command `\substitutefontfamily` creates an `.fd` file on the fly. The first argument is an encoding mnemonic, the second and third arguments are font family names.

```
\def\substitutefontfamily#1#2#3{%
  \lowercase{\immediate\openout15=#1#2.fd\relax}%
  \immediate\write15{%
    \string\ProvidesFile{#1#2.fd}%
    [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
     \space generated font description file]^^J
    \string\DeclareFontFamily{#1}{#2}{}^^J
    \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^^J
    \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^^J
    \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^^J
    \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^^J
    \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^^J
    \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^^J
    \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^^J
    \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^^J
    }%
  \closeout15
  }
```

This command should only be used in the preamble of a document.

```
\@onlypreamble\substitutefontfamily
```

```
⟨/package⟩
```

## 12 The Kernel of Babel

The kernel of the babel system is stored in either `hyphen.cfg` or `switch.def` and `babel.def`. The file `hyphen.cfg` is a file that can be loaded into the format, which is necessary when you want to be able to switch hyphenation patterns. The file `babel.def` contains some TEX code that can be read in at run time. When `babel.def` is loaded it checks if `hyphen.cfg` is in the format; if not the file `switch.def` is loaded.

Because plain TEX users might want to use some of the features of the babel system too, care has to be taken that plain TEX can process the files. For this reason the current format will have to be checked in a number of places. Some of the code below is common to plain TEX and LATEX, some of it is for the LATEX case only.

When the command `\AtBeginDocument` doesn't exist we assume that we are dealing with a plain-based format. In that case the file `plain.def` is needed.

```
⟨*kernel | core⟩
\ifx\AtBeginDocument\@undefined
```

But we need to use the second part of `plain.def` (when we load it from `switch.def`) which we can do by defining `\adddialect`.

```
⟨kernel&!patterns⟩   \def\adddialect{}
  \input plain.def\relax
\fi
⟨/kernel | core⟩
```

Check the presence of the command `\iflanguage`, if it is undefined read the file `switch.def`.

```
⟨*core⟩
\ifx\iflanguage\@undefined
  \input switch.def\relax
\fi
⟨/core⟩
```

## 12.1 Encoding issues (part 1)

The first thing we need to do is to determine, at `\begin{document}`, which latin fontencoding to use.

`\latinencoding` When text is being typeset in an encoding other than 'latin' (`OT1` or `T1`), it would be nice to still have Roman numerals come out in the Latin encoding. So we first assume that the current encoding at the end of processing the package is the Latin encoding.

```
⟨*core⟩
\AtEndOfPackage{\edef\latinencoding{\cf@encoding}}
```

But this might be overruled with a later loading of the package `fontenc`. Therefor we check at the execution of `\begin{document}` whether it was loaded with the `T1` option. The normal way to do this (using `\@ifpackageloaded`) is disabled for this package. Now we have to revert to parsing the internal macro `\@filelist` which contains all the filenames loaded.

```
\AtBeginDocument{%
  \gdef\latinencoding{OT1}%
  \ifx\cf@encoding\bbl@t@one
    \xdef\latinencoding{\bbl@t@one}%
  \else
    \@ifl@aded{def}{t1enc}{\xdef\latinencoding{\bbl@t@one}}{}%
  \fi
  }
```

`\latintext` Then we can define the command `\latintext` which is a declarative switch to a latin font-encoding.

```
\DeclareRobustCommand{\latintext}{%
  \fontencoding{\latinencoding}\selectfont
  \def\encodingdefault{\latinencoding}}
```

`\textlatin` This command takes an argument which is then typeset using the requested font encoding. In order to avoid many encoding switches it operates in a local scope.

```
\ifx\@undefined\DeclareTextFontCommand
  \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
\else
  \DeclareTextFontCommand{\textlatin}{\latintext}
\fi
⟨/core⟩
```

We also need to redefine a number of commands to ensure that the right font encoding is used, but this can't be done before `babel.def` is loaded.

## 12.2 Multiple languages

With TeX version 3.0 it has become possible to load hyphenation patterns for more than one language. This means that some extra administration has to be taken care of. The user has to know for which languages patterns have been loaded, and what values of `\language` have been used.

Some discussion has been going on in the TeX world about how to use `\language`. Some have suggested to set a fixed standard, i. e., patterns for each language should *always* be loaded in the same location. It has also been suggested to use the ISO list for this purpose. Others have pointed out that the ISO list contains more than 256 languages, which have *not* been numbered consecutively.

I think the best way to use `\language`, is to use it dynamically. This code implements an algorithm to do so. It uses an external file in which the person who maintains a TeX environment has to record for which languages he has hyphenation patterns *and* in which files these are stored[4]. When hyphenation exceptions are stored in a separate file this can be indicated by naming that file *after* the file with the hyphenation patterns.

This "configuration file" can contain empty lines and comments, as well as lines which start with an equals (=) sign. Such a line will instruct LaTeX that the hyphenation patterns just processed have to be known under an alternative name. Here is an example:

```
% File    : language.dat
% Purpose : tell iniTeX what files with patterns to load.
english    english.hyphenations
=british

dutch      hyphen.dutch exceptions.dutch % Nederlands
german hyphen.ger
```

As the file `switch.def` needs to be read only once, we check whether it was read before. If it was, the command `\iflanguage` is already defined, so we can stop processing.

```
⟨*kernel⟩
⟨*!patterns⟩
\expandafter\ifx\csname iflanguage\endcsname\relax \else
\expandafter\endinput
\fi
⟨/!patterns⟩
```

`\language` Plain TeX version 3.0 provides the primitive `\language` that is used to store the current language. When used with a pre-3.0 version this function has to be implemented by allocating a counter.

```
\ifx\language\@undefined
  \csname newcount\endcsname\language
\fi
```

`\last@language` Another counter is used to store the last language defined. For pre-3.0 formats an extra counter has to be allocated,

[4]This is because different operating systems sometimes use *very* different file-naming conventions.

```
\ifx\newlanguage\@undefined
  \csname newcount\endcsname\last@language
```

plain TeX version 3.0 uses `\count 19` for this purpose.

```
\else
  \countdef\last@language=19
\fi
```

`\addlanguage` To add languages to TeX's memory plain TeX version 3.0 supplies `\newlanguage`, in a pre-3.0 environment a similar macro has to be provided. For both cases a new macro is defined here, because the original `\newlanguage` was defined to be `\outer`.

For a format based on plain version 2.x, the definition of `\newlanguage` can not be copied because `\count 19` is used for other purposes in these formats. Therefor `\addlanguage` is defined using a definition based on the macros used to define `\newlanguage` in plain TeX version 3.0.

```
\ifx\newlanguage\@undefined
  \def\addlanguage#1{%
    \global\advance\last@language \@ne
    \ifnum\last@language<\@cclvi
    \else
        \errmessage{No room for a new \string\language!}%
    \fi
    \global\chardef#1\last@language
    \wlog{\string#1 = \string\language\the\last@language}}
```

For formats based on plain version 3.0 the definition of `\newlanguage` can be simply copied, removing `\outer`.

```
\else
  \def\addlanguage{\alloc@9\language\chardef\@cclvi}
\fi
```

`\adddialect` The macro `\adddialect` can be used to add the name of a dialect or variant language, for which an already defined hyphenation table can be used.

```
\def\adddialect#1#2{%
    \global\chardef#1#2\relax
    \wlog{\string#1 = a dialect from \string\language#2}}
```

`\iflanguage` Users might want to test (in a private package for instance) which language is currently active. For this we provide a test macro, `\iflanguage`, that has three arguments. It checks whether the first argument is a known language. If so, it compares the first argument with the value of `\language`. Then, depending on the result of the comparison, it executes either the second or the third argument.

```
\def\iflanguage#1{%
  \expandafter\ifx\csname l@#1\endcsname\relax
    \@nolanerr{#1}%
  \else
    \bbl@afterfi{\ifnum\csname l@#1\endcsname=\language
```

```
      \expandafter\@firstoftwo
    \else
      \expandafter\@secondoftwo
    \fi}%
  \fi}
```

\selectlanguage The macro `\selectlanguage` checks whether the language is already defined before it performs its actual task, which is to update `\language` and activate language-specific definitions.

To allow the call of `\selectlanguage` either with a control sequence name or with a simple string as argument, we have to use a trick to delete the optional escape character.

To convert a control sequence to a string, we use the `\string` primitive. Next we have to look at the first character of this string and compare it with the escape character. Because this escape character can be changed by setting the internal integer `\escapechar` to a character number, we have to compare this number with the character of the string. To do this we have to use TEX's backquote notation to specify the character as a number.

If the first character of the `\string`'ed argument is the current escape character, the comparison has stripped this character and the rest in the 'then' part consists of the rest of the control sequence name. Otherwise we know that either the argument is not a control sequence or `\escapechar` is set to a value outside of the character range 0–255.

If the user gives an empty argument, we provide a default argument for `\string`. This argument should expand to nothing.

```
\edef\selectlanguage{%
  \noexpand\protect
  \expandafter\noexpand\csname selectlanguage \endcsname
  }
```

Because the command `\selectlanguage` could be used in a moving argument it expands to `\protect\selectlanguage␣`. Therefor, we have to make sure that a macro `\protect` exists. If it doesn't it is `\let` to `\relax`.

```
\ifx\@undefined\protect\let\protect\relax\fi
```

As LATEX 2.09 writes to files *expanded* whereas LATEX $2_\varepsilon$ takes care *not* to expand the arguments of `\write` statements we need to be a bit clever about the way we add information to `.aux` files. Therefor we introduce the macro `\xstring` which should expand to the right amount of `\string`'s.

```
\ifx\documentclass\@undefined
  \def\xstring{\string\string\string}
\else
  \let\xstring\string
\fi
```

Since version 3.5 babel writes entries to the auxiliary files in order to typeset table of contents etc. in the correct language environment.

`\bbl@pop@language` *But* when the language change happens *inside* a group the end of the group doesn't write anything to the auxiliary files. Therefor we need TEX's `aftergroup` mechanism to help us. The command `\aftergroup` stores the token immediately following it to be executed when the current group is closed. So we define a temporary control sequence `\bbl@pop@language` to be executed at the end of the group. It calls `\bbl@set@language` with the name of the current language as its argument.

`\bbl@language@stack` The previous solution works for one level of nesting groups, but as soon as more levels are used it is no longer adequate. For that case we need to keep track of the nested languages using a stack mechanism. This stack is called `\bbl@language@stack` and initially empty.

```
\xdef\bbl@language@stack{}
```

When using a stack we need a mechanism to push an element on the stack and to retrieve the information afterwards.

`\bbl@push@language`
`\bbl@pop@language` The stack is simply a list of languagenames, separated with a '+' sign; the push function can be simple:

```
\def\bbl@push@language{%
  \xdef\bbl@language@stack{\languagename+\bbl@language@stack}%
  }
```

Retrieving information from the stack is a little bit less simple, as we need to remove the element from the stack while storing it in the macro `\languagename`. For this we first define a helper function.

`\bbl@pop@lang` This macro stores its first element (which is delimited by the '+'-sign) in `\languagename` and stores the rest of the string (delimited by '-') in its third argument.

```
\def\bbl@pop@lang#1+#2-#3{%
  \def\languagename{#1}\xdef#3{#2}%
  }
```

The reason for the somewhat weird arrangement of arguments to the helper function is the fact it is called in the following way:

```
\def\bbl@pop@language{%
  \expandafter\bbl@pop@lang\bbl@language@stack-\bbl@language@stack
```

This means that before `\bbl@pop@lang` is executed TEX first *expands* the stack, stored in `\bbl@language@stack`. The result of that is that the argument string of `\bbl@pop@lang` contains one or more language names, each followed by a '+'-sign (zero language names won't occur as this macro will only be called after something has been pushed on the stack) followed by the '-'-sign and finally the reference to the stack.

```
$$
  \expandafter\bbl@set@language\expandafter{\languagename}%
  }
```

Once the name of the previous language is retrieved from the stack, it is fed to `\bbl@set@language` to do the actual work of switching everything that needs switching.

```
\expandafter\def\csname selectlanguage \endcsname#1{%
  \bbl@push@language
  \aftergroup\bbl@pop@language
  \bbl@set@language{#1}}
```

`\bbl@set@language` The macro `\bbl@set@language` takes care of switching the language environment *and* of writing entries on the auxiliary files.

```
\def\bbl@set@language#1{%
  \edef\languagename{%
    \ifnum\escapechar=\expandafter`\string#1\@empty
    \else \string#1\@empty\fi}%
  \select@language{\languagename}%
```

We also write a command to change the current language in the auxiliary files.

```
  \if@filesw
    \protected@write\@auxout{}{\string\select@language{\languagename}}%
    \addtocontents{toc}{\xstring\select@language{\languagename}}%
    \addtocontents{lof}{\xstring\select@language{\languagename}}%
    \addtocontents{lot}{\xstring\select@language{\languagename}}%
  \fi}
```

First, check if the user asks for a known language. If so, update the value of `\language` and call `\originalTeX` to bring TeX in a certain pre-defined state.

```
\def\select@language#1{%
  \expandafter\ifx\csname l@#1\endcsname\relax
    \@nolanerr{#1}%
  \else
    \expandafter\ifx\csname date#1\endcsname\relax
      \@noopterr{#1}%
    \else
      \language=\csname l@#1\endcsname\relax
      \originalTeX
```

The name of the language is stored in the control sequence `\languagename`. The contents of this control sequence could be tested in the following way:

```
\edef\tmp{\string english}
\ifx\languagename\tmp
    ...
\else
    ...
\fi
```

The construction with `\string` is necessary because `\languagename` returns the name with characters of category code `12` (other). Then we have to *re*define

`\originalTeX` to compensate for the things that have been activated. To save memory space for the macro definition of `\originalTeX`, we construct the control sequence name for the `\noextras`⟨*lang*⟩ command at definition time by expanding the `\csname` primitive.

```
      \expandafter\def\expandafter\originalTeX
          \expandafter{\csname noextras#1\endcsname
                      \let\originalTeX\@empty}%
```

```
      \languageshorthands{none}%
      \babel@beginsave
```

Now activate the language-specific definitions. This is done by constructing the names of three macros by concatenating three words with the argument of `\selectlanguage`, and calling these macros.

```
      \csname captions#1\endcsname
      \csname date#1\endcsname
      \csname extras#1\endcsname\relax
```

The switching of the values of `\lefthyphenmin` and `\righthyphenmin` is somewhat different. First we save their current values, then we check if `\`⟨*lang*⟩`hyphenmins` is defined. If it is not, we set default values (2 and 3), otherwise the values in `\`⟨*lang*⟩`hyphenmins` will be used.

```
      \babel@savevariable\lefthyphenmin
      \babel@savevariable\righthyphenmin
      \expandafter\ifx\csname #1hyphenmins\endcsname\relax
        \set@hyphenmins\tw@\thr@@\relax
      \else
        \expandafter\expandafter\expandafter\set@hyphenmins
          \csname #1hyphenmins\endcsname\relax
      \fi
    \fi
  \fi}
```

`otherlanguage` The `otherlanguage` environment can be used as an alternative to using the `\selectlanguage` declarative command. When you are typesetting a document which mixes left-to-right and right-to-left typesetting you have to use this environment in order to let things work as you expect them to.

The first thing this environment does is store the name of the language in `\languagename`; it then calls `\selectlanguage␣` to switch on everything that is needed for this language The `\ignorespaces` command is necessary to hide the environment when it is entered in horizontal mode.

```
\long\def\otherlanguage#1{%
  \csname selectlanguage \endcsname{#1}%
  \ignorespaces
  }
```

The `\endotherlanguage` part of the environment calls `\originalTeX` to restore (most of) the settings and tries to hide itself when it is called in horizontal mode.

```
\long\def\endotherlanguage{%
```

```
  \originalTeX
  \global\@ignoretrue\ignorespaces
  }
```

otherlanguage* The otherlanguage environment is meant to be used when a large part of text from a different language needs to be typeset, but without changing the translation of words such as 'figure'.

This environment makes use of `\foreign@language`.

```
\expandafter\def\csname otherlanguage*\endcsname#1{%
  \foreign@language{#1}%
  }
```

At the end of the environment we need to switch off the extra definitions. The grouping mechanism of the environment will take care of resetting the correct hyphenation rules.

```
\expandafter\def\csname endotherlanguage*\endcsname{%
  \csname noextras\languagename\endcsname
  }
```

`\foreignlanguage` The `\foreignlanguage` command is another substitute for the `\selectlanguage` command. This command takes two arguments, the first argument is the name of the language to use for typesetting the text specified in the second argument.

Unlike `\selectlanguage` this command doesn't switch *everything*, it only switches the hyphenation rules and the extra definitions for the language specified. It does this within a group and assumes the `\extras`⟨*lang*⟩ command doesn't make any `\global` changes. The coding is very similar to part of `\selectlanguage`.

```
\def\foreignlanguage{\protect\csname foreignlanguage \endcsname}
\expandafter\def\csname foreignlanguage \endcsname#1#2{%
  \begingroup
    \originalTeX
    \foreign@language{#1}%
    #2%
    \csname noextras#1\endcsname
  \endgroup
  }
```

`\foreign@language` This macro does the work for `\foreignlanguage` and the otherlanguage* environment.

```
\def\foreign@language#1{%
```

First we need to store the name of the language and check that it is a known language.

```
  \def\languagename{#1}%
  \expandafter\ifx\csname l@#1\endcsname\relax
    \@nolanerr{#1}%
  \else
```

If it is we can select the proper hyphenation table and switch on the extra definitions for this language.

```
    \language=\csname l@#1\endcsname\relax
    \languageshorthands{none}%
```

Then we set the left- and right hyphenmin variables.

```
    \csname extras#1\endcsname
    \expandafter\ifx\csname #1hyphenmins\endcsname\relax
      \set@hyphenmins\tw@\thr@@\relax
    \else
      \expandafter\expandafter\expandafter\set@hyphenmins
        \csname #1hyphenmins\endcsname\relax
    \fi
  \fi
  }
```

`hyphenrules` The environment hyphenrules can be used to select *just* the hyphenation rules. This environment does *not* change `\languagename` and when the hyphenation rules specified were not loaded it has no effect.

```
\def\hyphenrules#1{%
  \expandafter\ifx\csname l@#1\endcsname\@undefined
    \@nolanerr{#1}%
  \else
    \language=\csname l@#1\endcsname\relax
    \languageshorthands{none}%
  \fi
  }
\def\endhyphenrules{}
```

`\providehyphenmins` The macro `\providehyphenmins` should be used in the language definition files to provide a *default* setting for the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`. If the macro `\`⟨*lang*⟩`hyphenmins` is already defined this command has no effect.

```
\def\providehyphenmins#1#2{%
  \expandafter\ifx\csname #1hyphenmins\endcsname\relax
    \@namedef{#1hyphenmins}{#2}%
  \fi}
```

`\set@hyphenmins` This macro sets the values of `\lefthyphenmin` and `\righthyphenmin`. It expects two values as its argument.

```
\def\set@hyphenmins#1#2{\lefthyphenmin#1\righthyphenmin#2}
```

`\LdfInit` This macro is defined in two versions. The first version is to be part of the 'kernel' of babel, ie. the part that is loaded in the format; the second version is defined in `babel.def`. The version in the format just checks the category code of the ampersand and then loads `babel.def`.

```
\def\LdfInit{%
  \chardef\atcatcode=\catcode`\@
  \catcode`\@=11\relax
  \input babel.def\relax
```

The category code of the ampersand is restored and the macro calls itself again with the new definition from `babel.def`

```
  \catcode`\@=\atcatcode \let\atcatcode\relax
  \LdfInit}
⟨/kernel⟩
```

The second version of this macro takes two arguments. The first argument is the name of the language that will be defined in the language definition file; the second argument is either a control sequence or a string from which a control sequence should be constructed. The existence of the control sequence indicates that the file has been processed before.

At the start of processing a language definition file we always check the category code of the ampersand. We make sure that it is a 'letter' during the processing of the file.

```
⟨*core⟩
\def\LdfInit#1#2{%
  \chardef\atcatcode=\catcode`\@
  \catcode`\@=11\relax
```

Another character that needs to have the correct category code during processing of language definition files is the equals sign, '=', because it is sometimes used in constructions with the `\let` primitive. Therefor we store its current catcode and restore it later on.

```
  \chardef\eqcatcode=\catcode`\=
  \catcode`\==12\relax
```

Now we check whether we should perhaps stop the processing of this file. To do this we first need to check whether the second argument that is passed to `\LdfInit` is a control sequence. We do that by looking at the first token after passing `#2` through `string`. When it is equal to `\@backslashchar` we are dealing with a control sequence which we can compare with `\@undefined`.

```
  \let\bbl@tempa\relax
  \expandafter\if\expandafter\@backslashchar
                  \expandafter\@car\string#2\@nil
    \ifx#2\@undefined
    \else
```

If so, we call `\ldf@quit` (but after the end of this `\if` construction) to set the main language, restore the category code of the @-sign and call `\endinput`.

```
      \def\bbl@tempa{\ldf@quit{#1}}
    \fi
  \else
```

When `#2` was *not* a control sequence we construct one and compare it with `\relax`.

```
    \expandafter\ifx\csname#2\endcsname\relax
    \else
      \def\bbl@tempa{\ldf@quit{#1}}
    \fi
  \fi
  \bbl@tempa
```

Finally we check `\originalTeX`.

```
  \ifx\originalTeX\@undefined
    \let\originalTeX\@empty
  \else
    \originalTeX
  \fi}
```

`\ldf@quit` This macro interrupts the processing of a language definition file.

```
\def\ldf@quit#1{%
  \expandafter\main@language\expandafter{#1}%
  \catcode`\@=\atcatcode \let\atcatcode\relax
  \catcode`\==\eqcatcode \let\eqcatcode\relax
  \endinput
}
```

`\ldf@finish` This macro takes one argument. It is the name of the language that was defined in the language definition file.

We load the local configuration file if one is present, we set the main language (taking into account that the argument might be a control sequence that needs to be expanded) and reset the category code of the @-sign.

```
\def\ldf@finish#1{%
  \loadlocalcfg{#1}%
  \expandafter\main@language\expandafter{#1}%
  \catcode`\@=\atcatcode \let\atcatcode\relax
  \catcode`\==\eqcatcode \let\eqcatcode\relax
  }
```

After the preamble of the document the commands `\LdfInit`, `\ldf@quit` and `\ldf@finish` are no longer needed. Therefor they are turned into warning messages in LaTeX.

```
\@onlypreamble\LdfInit
\@onlypreamble\ldf@quit
\@onlypreamble\ldf@finish
```

`\main@language` `\bbl@main@language` This command should be used in the various language definition files. It stores its argument in `\bbl@main@language`; to be used to switch to the correct language at the beginning of the document.

```
\def\main@language#1{%
  \def\bbl@main@language{#1}%
  \let\languagename\bbl@main@language
  \language=\csname l@\languagename\endcsname\relax
  }
```

The default is to use English as the main language.

```
\ifx\l@english\@undefined
  \let\l@english\z@
\fi
\main@language{english}
```

We also have to make sure that some code gets executed at the beginning of the document.

```
\AtBeginDocument{%
  \expandafter\selectlanguage\expandafter{\bbl@main@language}}
⟨/core⟩
```

`\originalTeX` The macro`\originalTeX` should be known to TeX at this moment. As it has to be expandable we `\let` it to `\@empty` instead of `\relax`.

```
⟨*kernel⟩
\ifx\originalTeX\@undefined\let\originalTeX\@empty\fi
```

Because this part of the code can be included in a format, we make sure that the macro which initialises the save mechanism, `\babel@beginsave`, is not considered to be undefined.

```
\ifx\babel@beginsave\@undefined\let\babel@beginsave\relax\fi
```

`\@nolanerr` `\@nopatterns` The babel package will signal an error when a documents tries to select a language that hasn't been defined earlier. When a user selects a language for which no hyphenation patterns were loaded into the format he will be given a warning about that fact. We revert to the patterns for `\language`=0 in that case. In most formats that will be (US)english, but it might also be empty.

`\@noopterr` When the package was loaded without options not everything will work as expected. An error message is issued in that case.

When the format knows about `\PackageError` it must be LaTeX 2$_\varepsilon$, so we can safely use its error handling interface. Otherwise we'll have to 'keep it simple'.

```
\ifx\PackageError\@undefined
  \def\@nolanerr#1{%
    \errhelp{Your command will be ignored, type <return> to proceed}%
    \errmessage{You haven't defined the language #1\space yet}}
  \def\@nopatterns#1{%
    \message{No hyphenation patterns were loaded for}%
    \message{the language `#1'}%
    \message{I will use the patterns loaded for \string\language=0
        instead}}
  \def\@noopterr#1{%
    \errmessage{The option #1 was not specified in \string\usepackage}
    \errhelp{You may continue, but expect unexpected results}}
  \def\@activated#1{%
    \wlog{Package babel Info: Making #1 an active character}}
\else
  \newcommand*{\@nolanerr}[1]{%
    \PackageError{babel}%
                 {You haven't defined the language #1\space yet}%
        {Your command will be ignored, type <return> to proceed}}
  \newcommand*{\@nopatterns}[1]{%
    \PackageWarningNoLine{babel}%
        {No hyphenation patterns were loaded for\MessageBreak
          the language `#1'\MessageBreak
```

```
              I will use the patterns loaded for \string\language=0
              instead}}
  \newcommand*{\@noopterr}[1]{%
    \PackageError{babel}%
                 {You haven't loaded the option #1\space yet}%
            {You may proceed, but expect unexpected results}}
  \newcommand*{\@activated}[1]{%
    \PackageInfo{babel}{%
      Making #1 an active character}}
\fi
```

The following code is meant to be read by iniTEX because it should instruct TEX to read hyphenation patterns. To this end the docstrip option patterns can be used to include this code in the file hyphen.cfg.

```
⟨*patterns⟩
```

\process@line Each line in the file language.dat is processed by \process@line after it is read. The first thing this macro does is to check whether the line starts with =. When the first token of a line is an =, the macro \process@synonym is called; otherwise the macro \process@language will continue.

```
\def\process@line#1#2 #3/{%
  \ifx=#1
    \process@synonym#2 /
  \else
    \process@language#1#2 #3/%
  \fi
  }
```

\process@synonym This macro takes care of the lines which start with an =. It needs an empty token register to begin with.

```
\toks@{}
\def\process@synonym#1 /{%
  \ifnum\last@language=\m@ne
```

When no languages have been loaded yet, the name following the = will be a synonym for hyphenation register 0.

```
    \expandafter\chardef\csname l@#1\endcsname0\relax
    \wlog{\string\l@#1=\string\language0}
```

As no hyphenation patterns are read in yet, we can not yet set the hyphenmin parameters. Therefor a commands to do so is stored in a token register and executed when the first pattern file has been processed.

```
    \toks@\expandafter{\the\toks@
      \expandafter\let\csname #1hyphenmins\expandafter\endcsname
      \csname\languagename hyphenmins\endcsname}%
  \else
```

Otherwise the name will be a synonym for the language loaded last.

```
    \expandafter\chardef\csname l@#1\endcsname\last@language
    \wlog{\string\l@#1=\string\language\the\last@language}
```

We also need to copy the hyphenmin parameters for the synonym.

```
      \expandafter\let\csname #1hyphenmins\expandafter\endcsname
      \csname\languagename hyphenmins\endcsname
    \fi
    }
```

\process@language The macro `\process@language` is used to process a non-empty line from the 'configuration file'. It has three arguments, each delimited by white space. The third argument is optional, so a `/` character is expected to delimit the last argument. The first argument is the 'name' of a language; the second is the name of the file that contains the patterns. The optional third argument is the name of a file containing hyphenation exceptions.

The first thing to do is call `\addlanguage` to allocate a pattern register and to make that register 'active'.

```
\def\process@language#1 #2 #3/{%
    \expandafter\addlanguage\csname l@#1\endcsname
    \expandafter\language\csname l@#1\endcsname
    \def\languagename{#1}%
```

Then the 'name' of the language that will be loaded now is added to the token register `\toks8`. and finally the pattern file is read.

```
    \global\toks8\expandafter{\the\toks8#1, }%
```

For some hyphenation patterns it is needed to load them with a specific font encoding selected. This can be specified in the file `language.dat` by adding for instance '`:T1`' to the name of the language. The macro `\bbl@get@enc` extracts the font encoding from the language name and stores it in `\bbl@hyph@enc`.

```
    \begingroup
      \bbl@get@enc#1:\@@@
      \ifx\bbl@hyph@enc\@empty
      \else
        \fontencoding{\bbl@hyph@enc}\selectfont
      \fi
```

Some pattern files contain assignments to `\lefthyphenmin` and `\righthyphenmin`. TEX does not keep track of these assignments. Therefor we try to detect such assignments and store them in the `\`⟨*lang*⟩`hyphenmins` macro. When no assignments were made we provide a default setting.

```
      \lefthyphenmin\m@ne
```

Some pattern files contain changes to the `\lccode` en `\uccode` arrays. Such changes should remain local to the language; therefor we process the pattern file in a group; the `\patterns` command acts globally so its effect will be remembered.

```
      \input #2\relax
```

Now we globally store the settings of `\lefthyphenmin` and `\righthyphenmin` and close the group.

```
      \ifnum\lefthyphenmin=\m@ne
      \else
      \expandafter\xdef\csname #1hyphenmins\endcsname{%
        \the\lefthyphenmin\the\righthyphenmin}%
    \fi
  \endgroup
```

If the counter `\language` is still equal to zero we set the hyphenmin parameters to the values for the language loaded on pattern register 0.

```
  \ifnum\the\language=\z@
    \expandafter\ifx\csname #1hyphenmins\endcsname\relax
      \set@hyphenmins\tw@\thr@@\relax
    \else
      \expandafter\expandafter\expandafter\set@hyphenmins
        \csname #1hyphenmins\endcsname
    \fi
```

Now execute the contents of token register zero as it may contain commands which set the hyphenmin parameters for synonyms that were defined before the first pattern file is read in.

```
    \the\toks@
  \fi
```

Empty the token register after use.

```
  \toks@{}%
```

When the hyphenation patterns have been processed we need to see if a file with hyphenation exceptions needs to be read. This is the case when the third argument is not empty and when it does not contain a space token.

```
  \def\bbl@tempa{#3}%
  \ifx\bbl@tempa\@empty
  \else
    \ifx\bbl@tempa\space
    \else
      \input #3\relax
    \fi
  \fi
  }
```

`\bbl@get@enc`
`\bbl@hyph@enc`

The macro `\bbl@get@enc` extracts the font encoding from the language name and stores it in `\bbl@hyph@enc`. It uses delimited arguments to achieve this.

```
\def\bbl@get@enc#1:#2\@@@{%
```

First store both arguments in temporary macros,

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

then, if the second argument was empty, no font encoding was specified and we're done.

```
  \ifx\bbl@tempb\@empty
    \let\bbl@hyph@enc\@empty
  \else
```

But if the second argument was *not* empty it will now have a superfluous colon attached to it which we need to remove. This done by feeding it to `\bbl@get@enc`. The string that we are after will then be in the first argument and be stored in `\bbl@tempa`.

```
    \bbl@get@enc#2\@@@
    \edef\bbl@hyph@enc{\bbl@tempa}%
  \fi}
```

`\readconfigfile` The configuration file can now be opened for reading.

```
\openin1 = language.dat
```

See if the file exists, if not, use the default hyphenation file `hyphen.tex`. The user will be informed about this.

```
\ifeof1
  \message{I couldn't find the file language.dat,\space
           I will try the file hyphen.tex}
  \input hyphen.tex\relax
\else
```

Pattern registers are allocated using count register `\last@language`. Its initial value is 0. The definition of the macro `\newlanguage` is such that it first increments the count register and then defines the language. In order to have the first patterns loaded in pattern register number 0 we initialize `\last@language` with the value −1.

```
  \last@language\m@ne
```

We now read lines from the file until the end is found

```
  \loop
```

While reading from the input, it is useful to switch off recognition of the end-of-line character. This saves us stripping off spaces from the contents of the control sequence.

```
    \endlinechar\m@ne
    \read1 to \bbl@line
    \endlinechar`\^^M
```

Empty lines are skipped.

```
    \ifx\bbl@line\@empty
    \else
```

Now we add a space and a `/` character to the end of `\bbl@line`. This is needed to be able to recognize the third, optional, argument of `\process@language` later on.

```
      \edef\bbl@line{\bbl@line\space/}%
      \expandafter\process@line\bbl@line
    \fi
```

Check for the end of the file. To avoid a new `if` control sequence we create the necessary `\iftrue` or `\iffalse` with the help of `\csname`. But there is one complication with this approach: when skipping the `loop...repeat` TEX has to read `\if`/`\fi` pairs. So we have to insert a 'dummy' `\iftrue`.

```
    \iftrue \csname fi\endcsname
    \csname if\ifeof1 false\else true\fi\endcsname
  \repeat
```

Reactivate the default patterns,

```
  \language=0
\fi
```

and close the configuration file.

```
\closein1
```

Also remove some macros from memory

```
\let\process@language\@undefined
\let\process@synonym\@undefined
\let\process@line\@undefined
\let\bbl@tempa\@undefined
\let\bbl@tempb\@undefined
\let\bbl@eq@\@undefined
\let\bbl@line\@undefined
\let\bbl@get@enc\@undefined
```

We add a message about the fact that babel is loaded in the format and with which language patterns to the `\everyjob` register.

```
\ifx\addto@hook\@undefined
\else
  \expandafter\addto@hook\expandafter\everyjob\expandafter{%
    \expandafter\typeout\expandafter{\the\toks8 loaded.}}
\fi
```

Here the code for iniTEX ends.

```
⟨/patterns⟩
⟨/kernel⟩
```

## 12.3 Support for active characters

`\bbl@add@special` The macro `\bbl@add@special` is used to add a new character (or single character control sequence) to the macro `\dospecials` (and `\@sanitize` if LATEX is used).

To keep all changes local, we begin a new group. Then we redefine the macros `\do` and `\@makeother` to add themselves and the given character without expansion.

```
⟨*core | shorthands⟩
\def\bbl@add@special#1{\begingroup
    \def\do{\noexpand\do\noexpand}%
    \def\@makeother{\noexpand\@makeother\noexpand}%
```

To add the character to the macros, we expand the original macros with the additional character inside the redefinition of the macros. Because `\@sanitize` can be undefined, we put the definition inside a conditional.

```
    \edef\x{\endgroup
```

```
      \def\noexpand\dospecials{\dospecials\do#1}%
      \expandafter\ifx\csname @sanitize\endcsname\relax \else
        \def\noexpand\@sanitize{\@sanitize\@makeother#1}%
      \fi}%
```

The macro `\x` contains at this moment the following:

`\endgroup\def\dospecials{`*old contents* `\do`⟨*char*⟩`}`.

If `\@sanitize` is defined, it contains an additional definition of this macro. The last thing we have to do, is the expansion of `\x`. Then `\endgroup` is executed, which restores the old meaning of `\x`, `\do` and `\@makeother`. After the group is closed, the new definition of `\dospecials` (and `\@sanitize`) is assigned.

```
  \x}
```

`\bbl@remove@special` The companion of the former macro is `\bbl@remove@special`. It is used to remove a character from the set macros `\dospecials` and `\@sanitize`.

To keep all changes local, we begin a new group. Then we define a help macro `\x`, which expands to empty if the characters match, otherwise it expands to its nonexpandable input. Because TeX inserts a `\relax`, if the corresponding `\else` or `\fi` is scanned before the comparison is evaluated, we provide a 'stop sign' which should expand to nothing.

```
\def\bbl@remove@special#1{\begingroup
    \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
                 \else\noexpand##1\noexpand##2\fi}%
```

With the help of this macro we define `\do` and `\make@other`.

```
    \def\do{\x\do}%
    \def\@makeother{\x\@makeother}%
```

The rest of the work is similar to `\bbl@add@special`.

```
    \edef\x{\endgroup
      \def\noexpand\dospecials{\dospecials}%
      \expandafter\ifx\csname @sanitize\endcsname\relax \else
        \def\noexpand\@sanitize{\@sanitize}%
      \fi}%
  \x}
```

## 12.4 Shorthands

`\initiate@active@char` A language definition file can call this macro to make a character active. This macro takes one argument, the character that is to be made active. When the character was already active this macro does nothing. Otherwise, this macro defines the control sequence `\normal@char`⟨*char*⟩ to expand to the character in its 'normal state' and it defines the active character to expand to `\normal@char`⟨*char*⟩ by default (⟨*char*⟩ being the character to be made active). Later its definition can be changed to expand to `\active@char`⟨*char*⟩ by calling `\bbl@activate{`⟨*char*⟩`}`.

For example, to make the double quote character active one could have the following line in a language definition file:

```
\initiate@active@char{"}
```

`\bbl@afterelse` `\bbl@afterfi` Because the code that is used in the handling of active characters may need to look ahead, we take extra care to 'throw' it over the `\else` and `\fi` parts of an `\if`-statement[5]. These macros will break if another `\if...\fi` statement appears in one of the arguments.

```
\long\def\bbl@afterelse#1\else#2\fi{\fi#1}
\long\def\bbl@afterfi#1\fi{\fi#1}
```

`\peek@token` To prevent error messages when a shorthand, which normally takes an argument, sees a `\par`, or `}`, or similar tokens, we need to be able to 'peek' at what is coming up next in the input stream. Depending on the category code of the token that is seen, we need to either continue the code for the active character, or insert the non-active version of that character in the output. The macro `\peek@token` therefore takes two arguments, with which it constructs the control sequence to expand next. It `\let`'s `\bbl@nexta` and `\bbl@nextb` to the two possible macros. This is necessary for `\bbl@test@token` to take the right decision.

```
%\def\peek@token#1#2{%
%  \expandafter\let\expandafter\bbl@nexta\csname #1\string#2\endcsname
%  \expandafter\let\expandafter\bbl@nextb
%    \csname system@active\string#2\endcsname
%  \futurelet\bbl@token\bbl@test@token}
```

`\bbl@test@token` When the result of peeking at the next token has yielded a token with category 'letter', 'other' or 'active' it is safe to proceed with evaluating the code for the shorthand. When a token is found with any other category code proceeding is unsafe and therefor the original shorthand character is inserted in the output. The macro that calls `\bbl@test@token` needs to setup `\bbl@nexta` and `\bbl@nextb` in order to achieve this.

```
%\def\bbl@test@token{%
%  \let\bbl@next\bbl@nexta
%  \ifcat\noexpand\bbl@token a%
%  \else
%    \ifcat\noexpand\bbl@token=%
%    \else
%      \ifcat\noexpand\bbl@token\noexpand\bbl@next
%      \else
%        \let\bbl@next\bbl@nextb
%      \fi
%    \fi
%  \fi
%  \bbl@next}
```

The macro `\initiate@active@char` takes all the necessary actions to make its argument a shorthand character. The real work is performed once for each character.

[5]This code is based on code presented in TUGboat vol. 12, no2, June 1991 in "An expansion Power Lemma" by Sonja Maus.

```
\def\initiate@active@char#1{%
  \expandafter\ifx\csname active@char\string##1\endcsname\relax
    \bbl@afterfi{\@initiate@active@char{#1}}%
  \fi}
```

Note that the definition of `\@initiate@active@char` needs an active character, for this the `~` is used. Some of the changes we need, do not have to become available later on, so we do it inside a group.

```
\begingroup
  \catcode`\~\active
  \def\x{\endgroup
    \def\@initiate@active@char##1{%
```

If the character is already active we provide the default expansion under this shorthand mechanism.

```
      \ifcat\noexpand##1\noexpand~\relax
        \@ifundefined{normal@char\string##1}{%
          \expandafter\let\csname normal@char\string##1\endcsname##1%
          \expandafter\gdef
            \expandafter##1%
            \expandafter{%
              \expandafter\active@prefix\expandafter##1%
              \csname normal@char\string##1\endcsname}}{}%
      \else
```

Otherwise we write a message in the transcript file,

```
        \@activated{##1}%
```

and define `\normal@char`⟨*char*⟩ to expand to the character in its default state.

```
        \@namedef{normal@char\string##1}{##1}%
```

If we are making the right quote active we need to change `\pr@m@s` as well.

```
        \ifx##1'%
          \let\prim@s\bbl@prim@s
```

Also, make sure that a single `'` in math mode 'does the right thing'.

```
          \@namedef{normal@char\string##1}{%
            \textormath{##1}{^\bgroup\prim@s}}%
        \fi
```

If we are using the caret as a shorthand character special care should be taken to make sure math still works. Therefor an extra level of expansion is introduced with a check for math mode on the upper level.

```
        \ifx##1^%
          \gdef\bbl@act@caret{%
            \ifmmode
              \csname normal@char\string^\endcsname
            \else
              \bbl@afterfi
              {\if@safe@actives
                \bbl@afterelse\csname normal@char\string##1\endcsname
               \else
          \bbl@afterfi\csname user@active\string##1\endcsname
        \fi}%
       \fi}
     \fi
```

To prevent problems with the loading of other packages after babel we reset the catcode of the character at the end of the package.

```
     \@ifpackagewith{babel}{KeepShorthandsActive}{}{%
      \edef\bbl@tempa{\catcode`\noexpand##1\the\catcode`##1}%
      \expandafter\AtEndOfPackage\expandafter{\bbl@tempa}}%
```

Now we set the lowercase code of the ~ equal to that of the character to be made active and execute the rest of the code inside a `\lowercase` 'environment'.

```
     \@tempcnta=\lccode`\~
     \lccode`~=`##1%
     \lowercase{%
```

Make the character active and add it to `\dospecials` and `\@sanitize`.

```
      \catcode`~\active
      \expandafter\bbl@add@special
       \csname \string##1\endcsname
```

Also re-activate it again at `\begin{document}`.

```
       \AtBeginDocument{%
        \catcode`##1\active
```

We also need to make sure that the shorthands are active during the processing of the `.aux` file. Otherwise some citations may give unexpected results in the printout when a shorthand was used in the optional argument of `\bibitem` for example.

```
        \if@filesw
         \immediate\write\@mainaux{%
          \string\catcode`##1\string\active}%
        \fi}%
```

Define the character to expand to

`\active@prefix` ⟨*char*⟩ `\normal@char`⟨*char*⟩

(where `\active@char`⟨*char*⟩ is *one* control sequence!).

```
      \expandafter\gdef
       \expandafter~%
       \expandafter{%
       \expandafter\active@prefix\expandafter##1%
       \csname normal@char\string##1\endcsname}}%
     \lccode`\~\@tempcnta
    \fi
```

For the active caret we first expand to `\bbl@act@caret` in order to be able to handle math mode correctly.

```
    \ifx##1^%
     \@namedef{active@char\string##1}{\bbl@act@caret}%
    \else
```

We define the first level expansion of `\active@char`⟨*char*⟩ to check the status of the `@safe@actives` flag. If it is set to true we expand to the 'normal' version of this character, otherwise we call `\@active@char`⟨*char*⟩.

```
      \@namedef{active@char\string##1}{%
        \if@safe@actives
          \bbl@afterelse\csname normal@char\string##1\endcsname
        \else
          \bbl@afterfi\csname user@active\string##1\endcsname
        \fi}%
    \fi
```

The next level of the code checks whether a user has defined a shorthand for himself with this character. First we check for a single character shorthand. If that doesn't exist we check for a shorthand with an argument.

```
    \@namedef{user@active\string##1}{%
      \expandafter\ifx
      \csname \user@group @sh@\string##1@\endcsname
      \relax
        \bbl@afterelse\bbl@sh@select\user@group##1%
      {user@active@arg\string##1}{language@active\string##1}%
      \else
        \bbl@afterfi\csname \user@group @sh@\string##1@\endcsname
      \fi}%
```

When there is also no user-level shorthand with an argument we will check whether there is a language defined shorthand for this active character. Before the next token is absorbed as argument we need to make sure that this is safe. Therefor `\peek@token` is called to decide that.

```
    \long\@namedef{user@active@arg\string##1}####1{%
      \expandafter\ifx
      \csname \user@group @sh@\string##1@\string####1@\endcsname
      \relax
        \bbl@afterelse
        \csname language@active\string##1\endcsname####1%
      \else
        \bbl@afterfi
        \csname \user@group @sh@\string##1@\string####1@%
        \endcsname
      \fi}%
```

In order to do the right thing when a shorthand with an argument is used by itself at the end of the line we provide a definition for the case of an empty argument. For that case we let the shorthand character expand to its non-active self.

```
    \@namedef{\user@group @sh@\string##1@@}{%
      \csname normal@char\string##1\endcsname}
```

Like the shorthands that can be defined by the user, a language definition file can also define shorthands with and without an argument, so we need two more macros to check if they exist.

```
    \@namedef{language@active\string##1}{%
```

```
      \expandafter\ifx
      \csname \language@group @sh@\string##1@\endcsname
      \relax
        \bbl@afterelse\bbl@sh@select\language@group##1%
        {language@active@arg\string##1}{system@active\string##1}%
      \else
        \bbl@afterfi
        \csname \language@group @sh@\string##1@\endcsname
      \fi}%

    \long\@namedef{language@active@arg\string##1}####1{%
      \expandafter\ifx
      \csname \language@group @sh@\string##1@\string####1@\endcsname
      \relax
        \bbl@afterelse
        \csname system@active\string##1\endcsname####1%
      \else
        \bbl@afterfi
        \csname \language@group @sh@\string##1@\string####1@%
        \endcsname
      \fi}%
```

And the same goes for the system level.

```
    \@namedef{system@active\string##1}{%
      \expandafter\ifx
      \csname \system@group @sh@\string##1@\endcsname
      \relax
        \bbl@afterelse\bbl@sh@select\system@group##1%
        {system@active@arg\string##1}{normal@char\string##1}%
      \else
        \bbl@afterfi\csname \system@group @sh@\string##1@\endcsname
      \fi}%
```

When no shorthands were found the 'normal' version of the active character is inserted.

```
    \long\@namedef{system@active@arg\string##1}####1{%
      \expandafter\ifx
      \csname \system@group @sh@\string##1@\string####1@\endcsname
      \relax
        \bbl@afterelse\csname normal@char\string##1\endcsname####1%
      \else
        \bbl@afterfi
        \csname \system@group @sh@\string##1@\string####1@\endcsname
      \fi}%
```

When a shorthand combination such as `''` ends up in a heading TEX would see `\protect'\protect'`. To prevent this from happening a shorthand needs to be defined at user level.

```
    \@namedef{user@sh@\string##1@\string\protect@}{%
      \csname user@active\string##1\endcsname}%
    }%
```

```
  }\x
```

`\bbl@sh@select` This command helps the shorthand supporting macros to select how to proceed. Note that this macro needs to be expandable as do all the shorthand macros in order for them to work in expansion-only environments such as the argument of `\hyphenation`.

This macro expects the name of a group of shorthands in its first argument and a shorthand character in its second argument. It will expand to either `\bbl@firstcs` or `\bbl@scndcs`. Hence two more arguments need to follow it.

```
\def\bbl@sh@select#1#2{%
  \expandafter\ifx\csname#1@sh@\string#2@sel\endcsname\relax
    \bbl@afterelse\bbl@scndcs
  \else
    \bbl@afterfi\csname#1@sh@\string#2@sel\endcsname
  \fi
}
```

`\active@prefix` The command `\active@prefix` which is used in the expansion of active characters has a function similar to `\OT1-cmd` in that it `\protect`s the active character whenever `\protect` is *not* `\@typeset@protect`.

```
\def\active@prefix#1{%
  \ifx\protect\@typeset@protect
  \else
```

When `\protect` is set to `\@unexpandable@protect` we make sure that the active character is als *not* expanded by inserting `\noexpand` in front of it. The `\@gobble` is needed to remove a token such as `\activechar:` (when the double colon was the active character to be dealt with).

```
    \ifx\protect\@unexpandable@protect
      \bbl@afterelse\bbl@afterfi\noexpand#1\@gobble
    \else
      \bbl@afterfi\bbl@afterfi\protect#1\@gobble
    \fi
  \fi}
```

`\if@safe@actives` In some circumstances it is necessary to be able to change the expansion of an active character on the fly. For this purpose the switch `@safe@actives` is available. The setting of this switch should be checked in the first level expansion of `\active@char`⟨*char*⟩.

```
\newif\if@safe@actives
\@safe@activesfalse
```

`\bbl@restore@actives` When the output routine kicks in while the active characters were made “safe” this must be undone in the headers to prevent unexpected typeset results. For this situation we define a command to make them “unsafe” again.

```
\def\bbl@restore@actives{\if@safe@actives\@safe@activesfalse\fi}
```

`\bbl@activate` This macro takes one argument, like `\initiate@active@char`. The macro is used to change the definition of an active character to expand to `\active@char`⟨*char*⟩ instead of `\normal@char`⟨*char*⟩.

```
\def\bbl@activate#1{%
  \expandafter\def
  \expandafter#1\expandafter{%
    \expandafter\active@prefix
    \expandafter#1\csname active@char\string#1\endcsname}%
}
```

`\bbl@deactivate` This macro takes one argument, like `\bbl@activate`. The macro doesn't really make a character non-active; it changes its definition to expand to `\normal@char`⟨*char*⟩.

```
\def\bbl@deactivate#1{%
  \expandafter\def
  \expandafter#1\expandafter{%
    \expandafter\active@prefix
    \expandafter#1\csname normal@char\string#1\endcsname}%
}
```

`\bbl@firstcs`
`\bbl@scndcs` These macros have two arguments. They use one of their arguments to build a control sequence from.

```
\def\bbl@firstcs#1#2{\csname#1\endcsname}
\def\bbl@scndcs#1#2{\csname#2\endcsname}
```

`\declare@shorthand` The command `\declare@shorthand` is used to declare a shorthand on a certain level. It takes three arguments:

1. a name for the collection of shorthands, i.e. 'system', or 'dutch';
2. the character (sequence) that makes up the shorthand, i.e. `~` or `"a`;
3. the code to be executed when the shorthand is encountered.

```
\def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
\def\@decl@short#1#2#3\@nil#4{%
  \def\bbl@tempa{#3}%
  \ifx\bbl@tempa\@empty
    \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
    \@namedef{#1@sh@\string#2@}{#4}%
  \else
    \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
    \@namedef{#1@sh@\string#2@\string#3@}{#4}%
  \fi}
```

`\textormath` Some of the shorthands that will be declared by the language definition files have to be usable in both text and mathmode. To achieve this the helper macro `\textormath` is provided.

```
\def\textormath#1#2{%
```

```
  \ifmmode
    \bbl@afterelse#2%
  \else
    \bbl@afterfi#1%
  \fi}
```

`\user@group` `\language@group` `\system@group` The current concept of 'shorthands' supports three levels or groups of shorthands. For each level the name of the level or group is stored in a macro. The default is to have a user group; use language group 'english' and have a system group called 'system'.

```
\def\user@group{user}
\def\language@group{english}
\def\system@group{system}
```

`\useshorthands` This is the user level command to tell LaTeX that user level shorthands will be used in the document. It takes one argument, the character that starts a shorthand.

```
\def\useshorthands#1{%
```

First note that this is user level.

```
  \def\user@group{user}%
```

Then initialize the character for use as a shorthand character.

```
  \initiate@active@char{#1}%
```

Now that TeX has seen the character its category code is fixed, but for the actions of `\bbl@activate` to succeed we need it to be active. Hence the trick with the `\lccode` to circumvent this.

```
  \@tempcnta\lccode`\~
  \lccode`~=`#1%
  \lowercase{\catcode`~\active\bbl@activate{~}}%
  \lccode`\~\@tempcnta}
```

`\defineshorthand` Currently we only support one group of user level shorthands, called 'user'.

```
\def\defineshorthand{\declare@shorthand{user}}
```

`\languageshorthands` A user level command to change the language from which shorthands are used.

```
\def\languageshorthands#1{\def\language@group{#1}}
```

`\aliasshorthand`

```
\def\aliasshorthand#1#2{%
```

First the new shorthand needs to be initialized,

```
  \expandafter\ifx\csname active@char\string#2\endcsname\relax
    \ifx\document\@notprerr
      \@notshorthand{#2}
    \else
      \initiate@active@char{#2}%
```

Then we need to use the `\lccode` trick to make the new shorthand behave like the old one. Therefore we save the current `\lccode` of the `~`-character and restore it later. Then we `\let` the new shorthand character be equal to the original.

```
      \@tempcnta\lccode`\~
      \lccode`~=`#2%
      \lowercase{\let~#1}%
      \lccode`\~\@tempcnta
    \fi
  \fi
}
```

`\@notshorthand`

```
\def\@notshorthand#1{%
      \PackageError{babel}{%
        The character `\string #1' should be made
        a shorthand character;\MessageBreak
        add the command \string\useshorthands\string{#1\string} to
        the preamble.\MessageBreak
        I will ignore your instruction}{}%
  }
```

`\shorthandon` `\shorthandoff` The first level definition of these macros just passes the argument on to `\bbl@switch@sh`, adding `\@nil` at the end to denote the end of the list of characters.

```
\newcommand*\shorthandon[1]{\bbl@switch@sh{on}#1\@nil}
\newcommand*\shorthandoff[1]{\bbl@switch@sh{off}#1\@nil}
```

`\bbl@switch@sh` The macro `\bbl@switch@sh` takes the list of characters apart one by one and subsequently switches the category code of the shorthand character according to the first argument of `\bbl@switch@sh`.

```
\def\bbl@switch@sh#1#2#3\@nil{%
```

But before any of this switching takes place we make sure that the character we are dealing with is known as a shorthand character. If it is, a macro such as `\active@char"` should exist.

```
  \@ifundefined{active@char\string#2}{%
    \PackageError{babel}{%
      The character '\string #2' is not a shorthand character
      in \languagename}{%
      Maybe you made a typing mistake?\MessageBreak
      I will ignore your instruction}}{%
    \csname bbl@switch@sh@#1\endcsname#2}%
```

Now that, as the first character in the list has been taken care of, we pass the rest of the list back to `\bbl@switch@sh`.

```
  \ifx#3\@empty\else
    \bbl@afterfi\bbl@switch@sh{#1}#3\@nil
  \fi}
```

`\bbl@switch@sh@off` All that is left to do is define the actual switching macros. Switching off is easy, we just set the category code to ‘other’ (12).

```
\def\bbl@switch@sh@off#1{\catcode`#112\relax}
```

`\bbl@switch@sh@on` But switching the shorthand character back on is a bit more tricky. It involves making sure that we have an active character to begin with when the macro is being defined. It also needs the use of `\lowercase` and `\lccode` trickery to get everything to work out as expected. And to keep things local that need to remain local a group is opened, which is closed as soon as `\x` gets executed.

```
\begingroup
  \catcode`\~\active
  \def\x{\endgroup
    \def\bbl@switch@sh@on##1{%
      \lccode`~=`##1%
      \lowercase{%
        \catcode`~\active
        }%
      }%
    }
```

The next operation makes the above definition effective.

```
\x
%
```

To prevent problems with constructs such as `\char"01A` when the double quote is made active, we define a shorthand on system level.

```
\declare@shorthand{system}{"}{\csname normal@char\string"\endcsname}
```

When the right quote is made active we need to take care of handling it correctly in mathmode. Therefore we define a shorthand at system level to make it expand to a non-active right quote in textmode, but expand to its original definition in mathmode. (Note that the right quote is ‘active’ in mathmode because of its mathcode.)

```
\declare@shorthand{system}{'}{%
  \textormath{\csname normal@char\string'\endcsname}%
             {\sp\bgroup\prim@s}}
```

When the left quote is made active we need to take care of handling it correctly when it is followed by for instance an open brace token. Therefore we define a shorthand at system level to make it expand to a non-active left quote.

```
\declare@shorthand{system}{`}{\csname normal@char\string`\endcsname}
```

`\bbl@prim@s`
`\bbl@pr@m@s` One of the internal macros that are involved in substituting `\prime` for each right quote in mathmode is `\prim@s`. This checks if the next character is a right quote. When the right quote is active, the definition of this macro needs to be adapted to look for an active right quote.

```
\def\bbl@prim@s{%
  \prime\futurelet\@let@token\bbl@pr@m@s}
```

```
\begingroup
  \catcode`\'\active\let'\relax
  \def\x{\endgroup
    \def\bbl@pr@m@s{%
      \ifx'\@let@token
        \expandafter\pr@@@s
      \else
        \ifx^\@let@token
          \expandafter\expandafter\expandafter\pr@@@t
        \else
          \egroup
        \fi
      \fi}%
    }
\x
```

```
⟨/core | shorthands⟩
```

Normally the ~ is active and expands to `\penalty\@M\␣`. When it is written to the `.aux` file it is written expanded. To prevent that and to be able to use the character ~ as a start character for a shorthand, it is redefined here as a one character shorthand on system level.

```
⟨*core⟩
\initiate@active@char{~}
\declare@shorthand{system}{~}{\leavevmode\nobreak\ }
\bbl@activate{~}
```

`\OT1dqpos` `\T1dqpos` The position of the double quote character is different for the OT1 and T1 encodings. It will later be selected using the `\f@encoding` macro. Therefor we define two macros here to store the position of the character in these encodings.

```
\expandafter\def\csname OT1dqpos\endcsname{127}
\expandafter\def\csname T1dqpos\endcsname{4}
```

When the macro `\f@encoding` is undefined (as it is in plain TeX) we define it here to expand to `OT1`

```
\ifx\f@encoding\@undefined
  \def\f@encoding{OT1}
\fi
```

## 12.5 Language attributes

Language attributes provide a means to give the user control over which features of the language definition files he wants to enable.

`\languageattribute` The macro `\languageattribute` checks whether its arguments are valid and then activates the selected language attribute.

```
\newcommand\languageattribute[2]{%
```

First check whether the language is known.

```
  \expandafter\ifx\csname l@#1\endcsname\relax
```

```
    \@nolanerr{#1}%
  \else
```

Than process each attribute in the list.

```
    \@for\bbl@attr:=#2\do{%
```

We want to make sure that each attribute is selected only once; therefor we store the already selected attributes in `\bbl@known@attribs`. When that control sequence is not yet defined this attribute is certainly not selected before.

```
      \ifx\bbl@known@attribs\@undefined
        \in@false
      \else
```

Now we need to see if the attribute occurs in the list of already selected attributes.

```
        \edef\bbl@tempa{\noexpand\in@{,#1-\bbl@attr,}%
          {,\bbl@known@attribs,}}%
        \bbl@tempa
      \fi
```

When the attribute was in the list we issue a warning; this might not be the users intention.

```
      \ifin@
        \PackageWarning{Babel}{%
          You have more than once selected the attribute
          '\bbl@attr'\MessageBreak for language #1}%
      \else
```

When we end up here the attribute is not selected before. So, we add it to the list of selected attributes and execute the associated TEX-code.

```
        \edef\bbl@tempa{%
          \noexpand\bbl@add@list\noexpand\bbl@known@attribs{#1-\bbl@attr}}%
        \bbl@tempa
        \edef\bbl@tempa{#1-\bbl@attr}%
        \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
        {\csname#1@attr@\bbl@attr\endcsname}%
        {\@attrerr{#1}{\bbl@attr}}%
     \fi
      }
  \fi}
```

This command should only be used in the preamble of a document.

```
\@onlypreamble\languageattribute
```

The error text to be issued when an unknown attribute is selected.

```
  \newcommand*{\@attrerr}[2]{%
    \PackageError{babel}%
                 {The attribute #2 is unknown for language #1.}%
        {Your command will be ignored, type <return> to proceed}}
```

`\bbl@declare@ttribute` This command adds the new language/attribute combination to the list of known attributes.

```
\def\bbl@declare@ttribute#1#2#3{%
  \bbl@add@list\bbl@attributes{#1-#2}%
```

Then it defines a control sequence to be executed when the attribute is used in a document. The result of this should be that the macro `\extras...` for the current language is extended, otherwise the attribute will not work as its code is removed from memory at `\begin{document}`.

```
  \expandafter\def\csname#1@attr@#2\endcsname{#3}%
  }
```

`\bbl@ifattributeset` This internal macro has 4 arguments. It can be used to interpret TeX code based on whether a certain attribute was set. This command should appear inside the argument to `\AtBeginDocument` because the attributes are set in the document preamble, *after* babel is loaded.

The first argument is the language, the second argument the attribute being checked, and the third and fourth arguments are the true and false clauses.

```
\def\bbl@ifattributeset#1#2#3#4{%
```

First we need to find out if any attributes were set; if not we're done.

```
  \ifx\bbl@known@attribs\@undefined
    \in@false
  \else
```

The we need to check the list of known attributes.

```
    \edef\bbl@tempa{\noexpand\in@{,#1-#2,}%
      {,\bbl@known@attribs,}}%
    \bbl@tempa
  \fi
```

When we're this far `\ifin@` has a value indicating if the attribute in question was set or not. Just to be safe the code to be executed is 'thrown over the `\fi`'.

```
  \ifin@
    \bbl@afterelse#3%
  \else
    \bbl@afterfi#4%
  \fi
  }
```

`\bbl@add@list` This internal macro adds its second argument to a comma separated list in its first argument. When the list is not defined yet (or empty), it will be initiated

```
\def\bbl@add@list#1#2{%
  \ifx#1\@undefined
    \def#1{#2}%
  \else
    \ifx#1\@empty
      \def#1{#2}%
    \else
      \edef#1{#1,#2}%
    \fi
  \fi
  }
```

\bbl@ifknown@ttrib An internal macro to check whether a given language/attribute is known. The macro takes 4 arguments, the language/attribute, the attribute list, the TeX-code to be executed when the attribute is known and the TeX-code to be executed otherwise.

```
\def\bbl@ifknown@ttrib#1#2{%
```

We first assume the attribute is unknown.

```
  \let\bbl@tempa\@secondoftwo
```

Then we loop over the list of known attributes, trying to find a match.

```
  \@for\bbl@tempb:=#2\do{%
    \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
    \ifin@
```

When a match is found the definition of `\bbl@tempa` is changed.

```
      \let\bbl@tempa\@firstoftwo
    \else
    \fi}%
```

Finally we execute `\bbl@tempa`.

```
  \bbl@tempa
}
```

\bbl@clear@ttribs This macro removes all the attribute code from LaTeX's memory at `\begin{document}` time (if any is present).

```
\def\bbl@clear@ttribs{%
  \ifx\bbl@attributes\@undefined\else
    \@for\bbl@tempa:=\bbl@attributes\do{%
      \expandafter\bbl@clear@ttrib\bbl@tempa.
      }%
    \let\bbl@attributes\@undefined
  \fi
  }
\def\bbl@clear@ttrib#1-#2.{%
  \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
\AtBeginDocument{\bbl@clear@ttribs}
```

## 12.6 Support for saving macro definitions

To save the meaning of control sequences using `\babel@save`, we use temporary control sequences. To save hash table entries for these control sequences, we don't use the name of the control sequence to be saved to construct the temporary name. Instead we simply use the value of a counter, which is reset to zero each time we begin to save new values. This works well because we release the saved meanings before we begin to save a new set of control sequence meanings (see `\selectlanguage` and `\originalTeX`).

\babel@savecnt
\babel@beginsave The initialization of a new save cycle: reset the counter to zero.

```
\def\babel@beginsave{\babel@savecnt\z@}
```

Before it's forgotten, allocate the counter and initialize all.

```
\newcount\babel@savecnt
\babel@beginsave
```

`\babel@save` The macro `\babel@save`⟨*csname*⟩ saves the current meaning of the control sequence ⟨*csname*⟩ to `\originalTeX`[6]. To do this, we let the current meaning to a temporary control sequence, the restore commands are appended to `\originalTeX` and the counter is incremented.

```
\def\babel@save#1{%
  \expandafter\let\csname babel@\number\babel@savecnt\endcsname #1\relax
  \begingroup
    \toks@\expandafter{\originalTeX \let#1=}%
    \edef\x{\endgroup
      \def\noexpand\originalTeX{\the\toks@ \expandafter\noexpand
         \csname babel@\number\babel@savecnt\endcsname\relax}}%
  \x
  \advance\babel@savecnt\@ne}
```

`\babel@savevariable` The macro `\babel@savevariable`⟨*variable*⟩ saves the value of the variable. ⟨*variable*⟩ can be anything allowed after the `\the` primitive.

```
\def\babel@savevariable#1{\begingroup
    \toks@\expandafter{\originalTeX #1=}%
    \edef\x{\endgroup
      \def\noexpand\originalTeX{\the\toks@ \the#1\relax}}%
  \x}
```

`\bbl@frenchspacing`
`\bbl@nonfrenchspacing` Some languages need to have `\frenchspacing` in effect. Others don't want that. The command `\bbl@frenchspacing` switches it on when it isn't already in effect and `\bbl@nonfrenchspacing` switches it off if necessary.

```
\def\bbl@frenchspacing{%
  \ifnum\the\sfcode`\.=\@m
    \let\bbl@nonfrenchspacing\relax
  \else
    \frenchspacing
    \let\bbl@nonfrenchspacing\nonfrenchspacing
  \fi}
\let\bbl@nonfrenchspacing\nonfrenchspacing
```

## 12.7 Support for extending macros

`\addto` For each language four control sequences have to be defined that control the language-specific definitions. To be able to add something to these macro once they have been defined the macro `\addto` is introduced. It takes two arguments, a ⟨*control sequence*⟩ and TEX-code to be added to the ⟨*control sequence*⟩.

If the ⟨*control sequence*⟩ has not been defined before it is defined now.

```
\def\addto#1#2{%
```

[6]`\originalTeX` has to be expandable, i. e. you shouldn't let it to `\relax`.

```
  \ifx#1\@undefined
    \def#1{#2}%
  \else
```

The control sequence could also expand to `\relax`, in which case a circular definition results. The net result is a stack overflow.

```
    \ifx#1\relax
      \def#1{#2}%
    \else
```

Otherwise the replacement text for the ⟨*control sequence*⟩ is expanded and stored in a token register, together with the TeX-code to be added. Finally the ⟨*control sequence*⟩ is *re*defined, using the contents of the token register.

```
      {\toks@\expandafter{#1#2}%
        \xdef#1{\the\toks@}}%
    \fi
  \fi
}
```

## 12.8 Macros common to a number of languages

`\allowhyphens` This macro makes hyphenation possible. Basically its definition is nothing more than `\nobreak \hskip 0pt plus 0pt`[7].

```
\def\bbl@t@one{T1}
\def\allowhyphens{%
  \ifx\cf@encoding\bbl@t@one\else\bbl@allowhyphens\fi}
\def\bbl@allowhyphens{\nobreak\hskip\z@skip}
```

`\set@low@box` The following macro is used to lower quotes to the same level as the comma. It prepares its argument in box register 0.

```
\def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
    \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
    \setbox\z@\hbox{\lower\dimen\z@ \box\z@}\ht\z@\ht\tw@ \dp\z@\dp\tw@}
```

`\save@sf@q` The macro `\save@sf@q` is used to save and reset the current space factor.

```
\def\save@sf@q #1{\leavevmode
 \begingroup
  \edef\@SF{\spacefactor \the\spacefactor}#1\@SF
 \endgroup
}
```

`\bbl@disc` For some languages the macro `\bbl@disc` is used to ease the insertion of discretionaries for letters that behave 'abnormally' at a breakpoint.

```
\def\bbl@disc#1#2{%
  \nobreak\discretionary{#2-}{}{#1}\allowhyphens}
```

[7]TeX begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.

## 12.9 Making glyphs available

The file `babel.dtx`[8] makes a number of glyphs available that either do not exist in the `OT1` encoding and have to be 'faked', or that are not accessible through `T1enc.def`.

## 12.10 Quotation marks

`\quotedblbase` In the `T1` encoding the opening double quote at the baseline is available as a separate character, accessible via `\quotedblbase`. In the `OT1` encoding it is not available, therefor we make it available by lowering the normal open quote character to the baseline.

```
\ProvideTextCommand{\quotedblbase}{OT1}{%
  \save@sf@q{\set@low@box{\textquotedblright\/}%
    \box\z@\kern-.04em\allowhyphens}}
```

Make sure that when an encoding other than `OT1` or `T1` is used this glyph can still be typeset.

```
\ProvideTextCommandDefault{\quotedblbase}{%
  \UseTextSymbol{OT1}{\quotedblbase}}
```

`\quotesinglbase` We also need the single quote character at the baseline.

```
\ProvideTextCommand{\quotesinglbase}{OT1}{%
  \save@sf@q{\set@low@box{\textquoteright\/}%
    \box\z@\kern-.04em\allowhyphens}}
```

Make sure that when an encoding other than `OT1` or `T1` is used this glyph can still be typeset.

```
\ProvideTextCommandDefault{\quotesinglbase}{%
  \UseTextSymbol{OT1}{\quotesinglbase}}
```

`\guillemotleft` `\guillemotright` The guillemet characters are not available in `OT1` encoding. They are faked.

```
\ProvideTextCommand{\guillemotleft}{OT1}{%
  \ifmmode
    \ll
  \else
    \save@sf@q{\nobreak
      \raise.2ex\hbox{$\scriptscriptstyle\ll$}\allowhyphens}%
  \fi}
\ProvideTextCommand{\guillemotright}{OT1}{%
  \ifmmode
    \gg
  \else
    \save@sf@q{\nobreak
      \raise.2ex\hbox{$\scriptscriptstyle\gg$}\allowhyphens}%
  \fi}
```

[8]The file described in this section has version number v3.8g, and was last revised on 2005/05/21.

Make sure that when an encoding other than `OT1` or `T1` is used these glyphs can still be typeset.

```
\ProvideTextCommandDefault{\guillemotleft}{%
  \UseTextSymbol{OT1}{\guillemotleft}}
\ProvideTextCommandDefault{\guillemotright}{%
  \UseTextSymbol{OT1}{\guillemotright}}
```

`\guilsinglleft` `\guilsinglright` The single guillemets are not available in `OT1` encoding. They are faked.

```
\ProvideTextCommand{\guilsinglleft}{OT1}{%
  \ifmmode
    <%
  \else
    \save@sf@q{\nobreak
      \raise.2ex\hbox{$\scriptscriptstyle<$}\allowhyphens}%
  \fi}
\ProvideTextCommand{\guilsinglright}{OT1}{%
  \ifmmode
    >%
  \else
    \save@sf@q{\nobreak
      \raise.2ex\hbox{$\scriptscriptstyle>$}\allowhyphens}%
  \fi}
```

Make sure that when an encoding other than `OT1` or `T1` is used these glyphs can still be typeset.

```
\ProvideTextCommandDefault{\guilsinglleft}{%
  \UseTextSymbol{OT1}{\guilsinglleft}}
\ProvideTextCommandDefault{\guilsinglright}{%
  \UseTextSymbol{OT1}{\guilsinglright}}
```

## 12.11 Letters

`\ij` `\IJ` The dutch language uses the letter 'ij'. It is available in `T1` encoded fonts, but not in the `OT1` encoded fonts. Therefor we fake it for the `OT1` encoding.

```
\DeclareTextCommand{\ij}{OT1}{%
  \allowhyphens i\kern-0.02em j\allowhyphens}
\DeclareTextCommand{\IJ}{OT1}{%
  \allowhyphens I\kern-0.02em J\allowhyphens}
\DeclareTextCommand{\ij}{T1}{\char188}
\DeclareTextCommand{\IJ}{T1}{\char156}
```

Make sure that when an encoding other than `OT1` or `T1` is used these glyphs can still be typeset.

```
\ProvideTextCommandDefault{\ij}{%
  \UseTextSymbol{OT1}{\ij}}
\ProvideTextCommandDefault{\IJ}{%
  \UseTextSymbol{OT1}{\IJ}}
```

`\dj` `\DJ` The croatian language needs the letters `\dj` and `\DJ`; they are available in the `T1` encoding, but not in the `OT1` encoding by default.

Some code to construct these glyphs for the OT1 encoding was made available to me by Stipcevic Mario, (stipcevic@olimp.irb.hr).

```
\def\crrtic@{\hrule height0.1ex width0.3em}
\def\crttic@{\hrule height0.1ex width0.33em}
%
\def\ddj@{%
  \setbox0\hbox{d}\dimen@=\ht0
  \advance\dimen@1ex
  \dimen@.45\dimen@
  \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
  \advance\dimen@ii.5ex
  \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
\def\DDJ@{%
  \setbox0\hbox{D}\dimen@=.55\ht0
  \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
  \advance\dimen@ii.15ex %           correction for the dash position
  \advance\dimen@ii-.15\fontdimen7\font %     correction for cmtt font
  \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
  \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
%
\DeclareTextCommand{\dj}{OT1}{\ddj@ d}
\DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

Make sure that when an encoding other than OT1 or T1 is used these glyphs can still be typeset.

```
\ProvideTextCommandDefault{\dj}{%
  \UseTextSymbol{OT1}{\dj}}
\ProvideTextCommandDefault{\DJ}{%
  \UseTextSymbol{OT1}{\DJ}}
```

\SS For the T1 encoding \SS is defined and selects a specific glyph from the font, but for other encodings it is not available. Therefor we make it available here.

```
\DeclareTextCommand{\SS}{OT1}{SS}
\ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}
```

## 12.12 Shorthands for quotation marks

Shorthands are provided for a number of different quotation marks, which make them usable both outside and inside mathmode.

\glq \grq The ‘german’ single quotes.

```
\ProvideTextCommand{\glq}{OT1}{%
  \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
\ProvideTextCommand{\glq}{T1}{%
  \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
\ProvideTextCommandDefault{\glq}{\UseTextSymbol{OT1}\glq}
```

The definition of \grq depends on the fontencoding. With T1 encoding no extra kerning is needed.

```
\ProvideTextCommand{\grq}{T1}{%
  \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
\ProvideTextCommand{\grq}{OT1}{%
  \save@sf@q{\kern-.0125em%
  \textormath{\textquoteleft}{\mbox{\textquoteleft}}%
  \kern.07em\relax}}
\ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

`\glqq` `\grqq` The ‘german’ double quotes.

```
\ProvideTextCommand{\glqq}{OT1}{%
  \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
\ProvideTextCommand{\glqq}{T1}{%
  \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
\ProvideTextCommandDefault{\glqq}{\UseTextSymbol{OT1}\glqq}
```

The definition of `\grqq` depends on the fontencoding. With `T1` encoding no extra kerning is needed.

```
\ProvideTextCommand{\grqq}{T1}{%
  \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
\ProvideTextCommand{\grqq}{OT1}{%
  \save@sf@q{\kern-.07em%
  \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}%
  \kern.07em\relax}}
\ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

`\flq` `\frq` The ‘french’ single guillemets.

```
\ProvideTextCommand{\flq}{OT1}{%
  \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
\ProvideTextCommand{\flq}{T1}{%
  \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
\ProvideTextCommandDefault{\flq}{\UseTextSymbol{OT1}\flq}
```

```
\ProvideTextCommand{\frq}{OT1}{%
  \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
\ProvideTextCommand{\frq}{T1}{%
  \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
\ProvideTextCommandDefault{\frq}{\UseTextSymbol{OT1}\frq}
```

`\flqq` `\frqq` The ‘french’ double guillemets.

```
\ProvideTextCommand{\flqq}{OT1}{%
  \textormath{\guillemotleft}{\mbox{\guillemotleft}}}
\ProvideTextCommand{\flqq}{T1}{%
  \textormath{\guillemotleft}{\mbox{\guillemotleft}}}
\ProvideTextCommandDefault{\flqq}{\UseTextSymbol{OT1}\flqq}
```

```
\ProvideTextCommand{\frqq}{OT1}{%
  \textormath{\guillemotright}{\mbox{\guillemotright}}}
\ProvideTextCommand{\frqq}{T1}{%
  \textormath{\guillemotright}{\mbox{\guillemotright}}}
\ProvideTextCommandDefault{\frqq}{\UseTextSymbol{OT1}\frqq}
```

## 12.13 Umlauts and trema's

The command `\"` needs to have a different effect for different languages. For German for instance, the 'umlaut' should be positioned lower than the default position for placing it over the letters a, o, u, A, O and U. When placed over an e, i, E or I it can retain its normal position. For Dutch the same glyph is always placed in the lower position.

`\umlauthigh` `\umlautlow` To be able to provide both positions of `\"` we provide two commands to switch the positioning, the default will be `\umlauthigh` (the normal positioning).

```
\def\umlauthigh{%
  \def\bbl@umlauta##1{\leavevmode\bgroup%
      \expandafter\accent\csname\f@encoding dqpos\endcsname
      ##1\allowhyphens\egroup}%
  \let\bbl@umlaute\bbl@umlauta}
\def\umlautlow{%
  \def\bbl@umlauta{\protect\lower@umlaut}}
\def\umlautelow{%
  \def\bbl@umlaute{\protect\lower@umlaut}}
\umlauthigh
```

`\lower@umlaut` The command `\lower@umlaut` is used to position the `\"` closer the the letter.

We want the umlaut character lowered, nearer to the letter. To do this we need an extra ⟨*dimen*⟩ register.

```
\expandafter\ifx\csname U@D\endcsname\relax
  \csname newdimen\endcsname\U@D
\fi
```

The following code fools TEX's `make_accent` procedure about the current x-height of the font to force another placement of the umlaut character.

```
\def\lower@umlaut#1{%
```

First we have to save the current x-height of the font, because we'll change this font dimension and this is always done globally.

```
  \leavevmode\bgroup
    \U@D 1ex%
```

Then we compute the new x-height in such a way that the umlaut character is lowered to the base character. The value of `.45ex` depends on the METAFONT parameters with which the fonts were built. (Just try out, which value will look best.)

```
    {\setbox\z@\hbox{%
      \expandafter\char\csname\f@encoding dqpos\endcsname}%
      \dimen@ -.45ex\advance\dimen@\ht\z@
```

If the new x-height is too low, it is not changed.

```
      \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
```

Finally we call the `\accent` primitive, reset the old x-height and insert the base character in the argument.

```
    \expandafter\accent\csname\f@encoding dqpos\endcsname
    \fontdimen5\font\U@D #1%
  \egroup}
```

For all vowels we declare `\"` to be a composite command which uses `\bbl@umlauta` or `\bbl@umlaute` to position the umlaut character. We need to be sure that these definitions override the ones that are provided when the package `fontenc` with option `OT1` is used. Therefor these declarations are postponed until the beginning of the document.

```
\AtBeginDocument{%
  \DeclareTextCompositeCommand{\"}{OT1}{a}{\bbl@umlauta{a}}%
  \DeclareTextCompositeCommand{\"}{OT1}{e}{\bbl@umlaute{e}}%
  \DeclareTextCompositeCommand{\"}{OT1}{i}{\bbl@umlaute{\i}}%
  \DeclareTextCompositeCommand{\"}{OT1}{\i}{\bbl@umlaute{\i}}%
  \DeclareTextCompositeCommand{\"}{OT1}{o}{\bbl@umlauta{o}}%
  \DeclareTextCompositeCommand{\"}{OT1}{u}{\bbl@umlauta{u}}%
  \DeclareTextCompositeCommand{\"}{OT1}{A}{\bbl@umlauta{A}}%
  \DeclareTextCompositeCommand{\"}{OT1}{E}{\bbl@umlaute{E}}%
  \DeclareTextCompositeCommand{\"}{OT1}{I}{\bbl@umlaute{I}}%
  \DeclareTextCompositeCommand{\"}{OT1}{O}{\bbl@umlauta{O}}%
  \DeclareTextCompositeCommand{\"}{OT1}{U}{\bbl@umlauta{U}}%
}
```

## 12.14 The redefinition of the style commands

The rest of the code in this file can only be processed by LaTeX, so we check the current format. If it is plain TeX, processing should stop here. But, because of the need to limit the scope of the definition of `\format`, a macro that is used locally in the following `\if` statement, this comparison is done inside a group. To prevent TeX from complaining about an unclosed group, the processing of the command `\endinput` is deferred until after the group is closed. This is accomplished by the command `\aftergroup`.

```
{\def\format{lplain}
\ifx\fmtname\format
\else
  \def\format{LaTeX2e}
  \ifx\fmtname\format
  \else
    \aftergroup\endinput
  \fi
\fi}
```

Now that we're sure that the code is seen by LaTeX only, we have to find out what the main (primary) document style is because we want to redefine some macros. This is only necessary for releases of LaTeX dated before December 1991. Therefor this part of the code can optionally be included in `babel.def` by specifying the `docstrip` option `names`.

```
⟨*names⟩
```

The standard styles can be distinguished by checking whether some macros are defined. In table 1 an overview is given of the macros that can be used for this purpose.

| | | |
|---|---|---|
| article | : | both the `\chapter` and `\opening` macros are undefined |
| report and book | : | the `\chapter` macro is defined and the `\opening` is undefined |
| letter | : | the `\chapter` macro is undefined and the `\opening` is defined |

Table 1: How to determine the main document style

The macros that have to be redefined for the `report` and `book` document styles happen to be the same, so there is no need to distinguish between those two styles.

`\doc@style` First a parameter `\doc@style` is defined to identify the current document style. This parameter might have been defined by a document style that already uses macros instead of hard-wired texts, such as `artikel1.sty` [6], so the existence of `\doc@style` is checked. If this macro is undefined, i. e., if the document style is unknown and could therefore contain hard-wired texts, `\doc@style` is defined to the default value '0'.

```
\ifx\@undefined\doc@style
  \def\doc@style{0}%
```

This parameter is defined in the following `if` construction (see table 1):

```
  \ifx\@undefined\opening
    \ifx\@undefined\chapter
      \def\doc@style{1}%
    \else
      \def\doc@style{2}%
    \fi
  \else
    \def\doc@style{3}%
  \fi%
\fi%
```

### 12.14.1 Redefinition of macros

Now here comes the real work: we start to redefine things and replace hard-wired texts by macros. These redefinitions should be carried out conditionally, in case it has already been done.

For the `figure` and `table` environments we have in all styles:

```
\@ifundefined{figurename}{\def\fnum@figure{\figurename{} \thefigure}}{}
\@ifundefined{tablename}{\def\fnum@table{\tablename{} \thetable}}{}
```

The rest of the macros have to be treated differently for each style. When `\doc@style` still has its default value nothing needs to be done.

```
\ifcase \doc@style\relax
\or
```

This means that `babel.def` is read after the `article` style, where no `\chapter` and `\opening` commands are defined[9].

First we have the `\tableofcontents`, `\listoffigures` and `\listoftables`:

```
\@ifundefined{contentsname}%
    {\def\tableofcontents{\section*{\contentsname\@mkboth
          {\uppercase{\contentsname}}{\uppercase{\contentsname}}}%
      \@starttoc{toc}}}{}

\@ifundefined{listfigurename}%
    {\def\listoffigures{\section*{\listfigurename\@mkboth
          {\uppercase{\listfigurename}}{\uppercase{\listfigurename}}}%
     \@starttoc{lof}}}{}

\@ifundefined{listtablename}%
    {\def\listoftables{\section*{\listtablename\@mkboth
          {\uppercase{\listtablename}}{\uppercase{\listtablename}}}%
      \@starttoc{lot}}}{}
```

Then the `\thebibliography` and `\theindex` environments.

```
\@ifundefined{refname}%
    {\def\thebibliography#1{\section*{\refname
      \@mkboth{\uppercase{\refname}}{\uppercase{\refname}}}%
      \list{[\arabic{enumi}]}{\settowidth\labelwidth{[#1]}%
        \leftmargin\labelwidth
        \advance\leftmargin\labelsep
        \usecounter{enumi}}%
        \def\newblock{\hskip.11em plus.33em minus.07em}%
        \sloppy\clubpenalty4000\widowpenalty\clubpenalty
        \sfcode`\.=1000\relax}}{}

\@ifundefined{indexname}%
    {\def\theindex{\@restonecoltrue\if@twocolumn\@restonecolfalse\fi
     \columnseprule \z@
     \columnsep 35pt\twocolumn[\section*{\indexname}]%
       \@mkboth{\uppercase{\indexname}}{\uppercase{\indexname}}%
       \thispagestyle{plain}%
       \parskip\z@ plus.3pt\parindent\z@\let\item\@idxitem}}{}
```

The `abstract` environment:

```
\@ifundefined{abstractname}%
    {\def\abstract{\if@twocolumn
    \section*{\abstractname}%
```

[9]A fact that was pointed out to me by Nico Poppelier and was already used in Piet van Oostrum's document style option `nl`.

```
    \else \small
    \begin{center}%
    {\bf \abstractname\vspace{-.5em}\vspace{\z@}}%
    \end{center}%
    \quotation
    \fi}{}
```

And last but not least, the macro `\part`:

```
\@ifundefined{partname}%
{\def\@part[#1]#2{\ifnum \c@secnumdepth >\m@ne
        \refstepcounter{part}%
        \addcontentsline{toc}{part}{\thepart
        \hspace{1em}#1}\else
      \addcontentsline{toc}{part}{#1}\fi
   {\parindent\z@ \raggedright
    \ifnum \c@secnumdepth >\m@ne
      \Large \bf \partname{} \thepart
      \par \nobreak
    \fi
    \huge \bf
    #2\markboth{}{}\par}%
    \nobreak
    \vskip 3ex\@afterheading}%
}{}
```

This is all that needs to be done for the `article` style.

```
\or
```

The next case is formed by the two styles `book` and `report`. Basically we have to do the same as for the `article` style, except now we must also change the `\chapter` command.

The tables of contents, figures and tables:

```
\@ifundefined{contentsname}%
    {\def\tableofcontents{\@restonecolfalse
      \if@twocolumn\@restonecoltrue\onecolumn
      \fi\chapter*{\contentsname\@mkboth
          {\uppercase{\contentsname}}{\uppercase{\contentsname}}}%
      \@starttoc{toc}%
      \csname if@restonecol\endcsname\twocolumn
      \csname fi\endcsname}}{}

\@ifundefined{listfigurename}%
    {\def\listoffigures{\@restonecolfalse
      \if@twocolumn\@restonecoltrue\onecolumn
      \fi\chapter*{\listfigurename\@mkboth
          {\uppercase{\listfigurename}}{\uppercase{\listfigurename}}}%
      \@starttoc{lof}%
      \csname if@restonecol\endcsname\twocolumn
      \csname fi\endcsname}}{}

```

```
\@ifundefined{listtablename}%
    {\def\listoftables{\@restonecolfalse
      \if@twocolumn\@restonecoltrue\onecolumn
      \fi\chapter*{\listtablename\@mkboth
          {\uppercase{\listtablename}}{\uppercase{\listtablename}}}%
      \@starttoc{lot}%
      \csname if@restonecol\endcsname\twocolumn
      \csname fi\endcsname}}{}
```

Again, the `bibliography` and `index` environments; notice that in this case we use `\bibname` instead of `\refname` as in the definitions for the `article` style. The reason for this is that in the `article` document style the term 'References' is used in the definition of `\thebibliography`. In the `report` and `book` document styles the term 'Bibliography' is used.

```
\@ifundefined{bibname}%
    {\def\thebibliography#1{\chapter*{\bibname
     \@mkboth{\uppercase{\bibname}}{\uppercase{\bibname}}}%
     \list{[\arabic{enumi}]}{\settowidth\labelwidth{[#1]}%
     \leftmargin\labelwidth \advance\leftmargin\labelsep
     \usecounter{enumi}}%
     \def\newblock{\hskip.11em plus.33em minus.07em}%
     \sloppy\clubpenalty4000\widowpenalty\clubpenalty
     \sfcode`\.=1000\relax}}{}

\@ifundefined{indexname}%
    {\def\theindex{\@restonecoltrue\if@twocolumn\@restonecolfalse\fi
    \columnseprule \z@
    \columnsep 35pt\twocolumn[\@makeschapterhead{\indexname}]%
      \@mkboth{\uppercase{\indexname}}{\uppercase{\indexname}}%
    \thispagestyle{plain}%
    \parskip\z@ plus.3pt\parindent\z@ \let\item\@idxitem}}{}
```

Here is the `abstract` environment:

```
\@ifundefined{abstractname}%
    {\def\abstract{\titlepage
    \null\vfil
    \begin{center}%
    {\bf \abstractname}%
    \end{center}}}{}
```

And last but not least the `\chapter`, `\appendix` and `\part` macros.

```
\@ifundefined{chaptername}{\def\@chapapp{\chaptername}}{}
%
\@ifundefined{appendixname}%
    {\def\appendix{\par
      \setcounter{chapter}{0}%
      \setcounter{section}{0}%
      \def\@chapapp{\appendixname}%
      \def\thechapter{\Alph{chapter}}}}{}
%
\@ifundefined{partname}%
```

```
    {\def\@part[#1]#2{\ifnum \c@secnumdepth >-2\relax
          \refstepcounter{part}%
          \addcontentsline{toc}{part}{\thepart
          \hspace{1em}#1}\else
          \addcontentsline{toc}{part}{#1}\fi
      \markboth{}{}%
      {\centering
       \ifnum \c@secnumdepth >-2\relax
         \huge\bf \partname{} \thepart
       \par
       \vskip 20pt \fi
       \Huge \bf
       #1\par}\@endpart}}{}%
\or
```

Now we address the case where `babel.def` is read after the `letter` style. The `letter` document style defines the macro `\opening` and some other macros that are specific to `letter`. This means that we have to redefine other macros, compared to the previous two cases.

First two macros for the material at the end of a letter, the `\cc` and `\encl` macros.

```
\@ifundefined{ccname}%
    {\def\cc#1{\par\noindent
     \parbox[t]{\textwidth}%
     {\@hangfrom{\rm \ccname : }\ignorespaces #1\strut}\par}}{}

\@ifundefined{enclname}%
    {\def\encl#1{\par\noindent
     \parbox[t]{\textwidth}%
     {\@hangfrom{\rm \enclname : }\ignorespaces #1\strut}\par}}{}
```

The last thing we have to do here is to redefine the `headings` pagestyle:

```
\@ifundefined{headtoname}%
    {\def\ps@headings{%
        \def\@oddhead{\sl \headtoname{} \ignorespaces\toname \hfil
                      \@date \hfil \pagename{} \thepage}%
        \def\@oddfoot{}}}{}
```

This was the last of the four standard document styles, so if `\doc@style` has another value we do nothing and just close the `if` construction.

```
\fi
```

Here ends the code that can be optionally included when a version of LaTeX is in use that is dated *before* December 1991.

```
⟨/names⟩
⟨/core⟩
```

## 12.15 Cross referencing macros

The LaTeX book states:

The *key* argument is any sequence of letters, digits, and punctuation symbols; upper- and lowercase letters are regarded as different.

When the above quote should still be true when a document is typeset in a language that has active characters, special care has to be taken of the category codes of these characters when they appear in an argument of the cross referencing macros.

When a cross referencing command processes its argument, all tokens in this argument should be character tokens with category 'letter' or 'other'.

The only way to accomplish this in most cases is to use the trick described in the TEXbook [1] (Appendix D, page 382). The primitive `\meaning` applied to a token expands to the current meaning of this token. For example, '`\meaning\A`' with `\A` defined as '`\def\A#1{\B}`' expands to the characters '`macro:#1->\B`' with all category codes set to 'other' or 'space'.

`\bbl@redefine` To redefine a command, we save the old meaning of the macro. Then we redefine it to call the original macro with the 'sanitized' argument. The reason why we do it this way is that we don't want to redefine the LATEX macros completely in case their definitions change (they have changed in the past).

Because we need to redefine a number of commands we define the command `\bbl@redefine` which takes care of this. It creates a new control sequence, `\org@...`

```
⟨*core | shorthands⟩
\def\bbl@redefine#1{%
  \edef\bbl@tempa{\expandafter\@gobble\string#1}%
  \expandafter\let\csname org@\bbl@tempa\endcsname#1
  \expandafter\def\csname\bbl@tempa\endcsname}
```

This command should only be used in the preamble of the document.

```
\@onlypreamble\bbl@redefine
```

`\bbl@redefine@long` This version of `\babel@redefine` can be used to redefine `\long` commands such as `\ifthenelse`.

```
\def\bbl@redefine@long#1{%
  \edef\bbl@tempa{\expandafter\@gobble\string#1}%
  \expandafter\let\csname org@\bbl@tempa\endcsname#1
  \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname}
\@onlypreamble\bbl@redefine@long
```

`\bbl@redefinerobust` For commands that are redefined, but which *might* be robust we need a slightly more intelligent macro. A robust command `foo` is defined to expand to `\protect\foo␣`. So it is necessary to check whether `\foo␣` exists.

```
\def\bbl@redefinerobust#1{%
  \edef\bbl@tempa{\expandafter\@gobble\string#1}%
  \expandafter\ifx\csname \bbl@tempa\space\endcsname\relax
    \expandafter\let\csname org@\bbl@tempa\endcsname#1
    \expandafter\edef\csname\bbl@tempa\endcsname{\noexpand\protect
      \expandafter\noexpand\csname\bbl@tempa\space\endcsname}%
  \else
    \expandafter\let\csname org@\bbl@tempa\expandafter\endcsname
                      \csname\bbl@tempa\space\endcsname
  \fi
```

The result of the code above is that the command that is being redefined is always robust afterwards. Therefor all we need to do now is define `\foo␣`.

```
  \expandafter\def\csname\bbl@tempa\space\endcsname}
```

This command should only be used in the preamble of the document.

```
\@onlypreamble\bbl@redefinerobust
```

`\newlabel` The macro `\label` writes a line with a `\newlabel` command into the `.aux` file to define labels.

```
%\bbl@redefine\newlabel#1#2{%
%  \@safe@activestrue\org@newlabel{#1}{#2}\@safe@activesfalse}
```

`\@newl@bel` We need to change the definition of the LaTeX-internal macro `\@newl@bel`. This is needed because we need to make sure that shorthand characters expand to their non-active version.

```
\def\@newl@bel#1#2#3{%
```

First we open a new group to keep the changed setting of `\protect` local and then we set the `@safe@actives` switch to true to make sure that any shorthand that appears in any of the arguments immediately expands to its non-active self.

```
  {%
    \@safe@activestrue
    \@ifundefined{#1@#2}%
      \relax
      {%
        \gdef \@multiplelabels {%
          \@latex@warning@no@line{There were multiply-defined labels}}%
        \@latex@warning@no@line{Label ‘#2’ multiply defined}%
      }%
    \global\@namedef{#1@#2}{#3}%
    }%
  }
```

`\@testdef` An internal LaTeX macro used to test if the labels that have been written on the `.aux` file have changed. It is called by the `\enddocument` macro. This macro needs to be completely rewritten, using `\meaning`. The reason for this is that in some cases the expansion of `\#1@#2` contains the same characters as the `#3`; but the character codes differ. Therefor LaTeX keeps reporting that the labels may have changed.

```
\CheckCommand*\@testdef[3]{%
  \def\reserved@a{#3}%
  \expandafter \ifx \csname #1@#2\endcsname \reserved@a
  \else
    \@tempswatrue
  \fi}
```

Now that we made sure that `\@testdef` still has the same definition we can rewrite it. First we make the shorthands 'safe'.

```
\def\@testdef #1#2#3{%
  \@safe@activestrue
```

Then we use `\bbl@tempa` as an 'alias' for the macro that contains the label which is being checked.

```
  \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
```

Then we define `\bbl@tempb` just as `\@newl@bel` does it.

```
  \def\bbl@tempb{#3}%
  \@safe@activesfalse
```

When the label is defined we replace the definition of `\bbl@tempa` by its meaning.

```
  \ifx\bbl@tempa\relax
  \else
    \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
  \fi
```

We do the same for `\bbl@tempb`.

```
  \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
```

If the label didn't change, `\bbl@tempa` and `\bbl@tempb` should be identical macros.

```
  \ifx \bbl@tempa \bbl@tempb
  \else
    \@tempswatrue
  \fi}
```

`\ref` `\pageref` The same holds for the macro `\ref` that references a label and `\pageref` to reference a page. So we redefine `\ref` and `\pageref`. While we change these macros, we make them robust as well (if they weren't already) to prevent problems if they should become expanded at the wrong moment.

```
\bbl@redefinerobust\ref#1{%
  \@safe@activestrue\org@ref{#1}\@safe@activesfalse}
\bbl@redefinerobust\pageref#1{%
  \@safe@activestrue\org@pageref{#1}\@safe@activesfalse}
```

`\@citex` The macro used to cite from a bibliography, `\cite`, uses an internal macro, `\@citex`. It is this internal macro that picks up the argument(s), so we redefine this internal macro and leave `\cite` alone. The first argument is used for typesetting, so the shorthands need only be deactivated in the second argument.

```
\bbl@redefine\@citex[#1]#2{%
  \@safe@activestrue\edef\@tempa{#2}\@safe@activesfalse
  \org@@citex[#1]{\@tempa}}
```

Unfortunately, the packages `natbib` and `cite` need a different definition of `\@citex`... To begin with, `natbib` has a definition for `\@citex` with *three* arguments... We only know that a package is loaded when `\begin{document}` is executed, so we need to postpone the different redefinition.

```
\AtBeginDocument{%
  \@ifpackageloaded{natbib}{%
```

Notice that we use `\def` here instead of `\bbl@redefine` because `\org@@citex` is already defined and we don't want to overwrite that definition (it would result in parameter stack overflow because of a circular definition).

```
    \def\@citex[#1][#2]#3{%
      \@safe@activestrue\edef\@tempa{#3}\@safe@activesfalse
      \org@@citex[#1][#2]{\@tempa}}%
  }{}}
```

The package `cite` has a definition of `\@citex` where the shorthands need to be turned off in both arguments.

```
\AtBeginDocument{%
  \@ifpackageloaded{cite}{%
    \def\@citex[#1]#2{%
      \@safe@activestrue\org@@citex[#1]{#2}\@safe@activesfalse}%
    }{}}
```

`\nocite` The macro `\nocite` which is used to instruct BiBTEX to extract uncited references from the database.

```
\bbl@redefine\nocite#1{%
  \@safe@activestrue\org@nocite{#1}\@safe@activesfalse}
```

`\bibcite` The macro that is used in the `.aux` file to define citation labels. When packages such as `natbib` or `cite` are not loaded its second argument is used to typeset the citation label. In that case, this second argument can contain active characters but is used in an environment where `\@safe@activestrue` is in effect. This switch needs to be reset inside the `\hbox` which contains the citation label. In order to determine during `.aux` file processing which definition of `\bibcite` is needed we define `\bibcite` in such a way that it redefines itself with the proper definition.

```
\bbl@redefine\bibcite{%
```

We call `\bbl@cite@choice` to select the proper definition for `\bibcite`. This new definition is then activated.

```
  \bbl@cite@choice
  \bibcite}
```

`\bbl@bibcite` The macro `\bbl@bibcite` holds the definition of `\bibcite` needed when neither `natbib` nor `cite` is loaded.

```
\def\bbl@bibcite#1#2{%
  \org@bibcite{#1}{\@safe@activesfalse#2}}
```

`\bbl@cite@choice` The macro `\bbl@cite@choice` determines which definition of `\bibcite` is needed.

```
\def\bbl@cite@choice{%
```

First we give `\bibcite` its default definition.

```
  \global\let\bibcite\bbl@bibcite
```

Then, when `natbib` is loaded we restore the original definition of `\bibcite` .

```
  \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
```

For `cite` we do the same.

```
12.1259   \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
```

Make sure this only happens once.

```
12.1260   \global\let\bbl@cite@choice\relax
12.1261   }
```

When a document is run for the first time, no `.aux` file is available, and `\bibcite` will not yet be properly defined. In this case, this has to happen before the document starts.

```
12.1262 \AtBeginDocument{\bbl@cite@choice}
```

`\@bibitem` One of the two internal LaTeX macros called by `\bibitem` that write the citation label on the `.aux` file.

```
12.1263 \bbl@redefine\@bibitem#1{%
12.1264   \@safe@activestrue\org@@bibitem{#1}\@safe@activesfalse}
```

## 12.16 marks

`\markright` `\markboth` Because the output routine is asynchronous, we must pass the current language attribute to the head lines, together with the text that is put into them. To achieve this we need to adapt the definition of `\markright` and `\markboth` somewhat.

```
12.1265 \bbl@redefine\markright#1{%
```

First of all we temporarily store the language switching command, using an expanded definition in order to get the current value of `\languagename`.

```
12.1266   \edef\bbl@tempb{\noexpand\protect
12.1267     \noexpand\foreignlanguage{\languagename}}%
```

Then, we check whether the argument is empty; if it is, we just make sure the scratch token register is empty.

```
12.1268   \def\bbl@arg{#1}%
12.1269   \ifx\bbl@arg\@empty
12.1270     \toks@{}%
12.1271   \else
```

Next, we store the argument to `\markright` in the scratch token register, together with the expansion of `\bbl@tempb` (containing the language switching command) as defined before. This way these commands will not be expanded by using `\edef` later on, and we make sure that the text is typeset using the correct language settings. While doing so, we make sure that active characters that may end up in the mark are not disabled by the output routine kicking in while `\@safe@activestrue` is in effect.

```
12.1272     \expandafter\toks@\expandafter{%
12.1273               \bbl@tempb{\protect\bbl@restore@actives#1}}%
12.1274   \fi
```

Then we define a temporary control sequence using `\edef`.

```
12.1275   \edef\bbl@tempa{%
```

When `\bbl@tempa` is executed, only `\languagename` will be expanded, because of the way the token register was filled.

```
    \noexpand\org@markright{\the\toks@}}%
  \bbl@tempa
}
```

The definition of `\markboth` is equivalent to that of `\markright`, except that we need two token registers.

```
\bbl@redefine\markboth#1#2{%
  \edef\bbl@tempb{\noexpand\protect
    \noexpand\foreignlanguage{\languagename}}%
  \def\bbl@arg{#1}%
  \ifx\bbl@arg\@empty
    \toks@{}%
  \else
   \expandafter\toks@\expandafter{%
             \bbl@tempb{\protect\bbl@restore@actives#1}}%
  \fi
  \def\bbl@arg{#2}%
  \ifx\bbl@arg\@empty
    \toks8{}%
  \else
    \expandafter\toks8\expandafter{%
             \bbl@tempb{\protect\bbl@restore@actives#2}}%
  \fi
  \edef\bbl@tempa{%
    \noexpand\org@markboth{\the\toks@}{\the\toks8}}%
  \bbl@tempa
}
⟨/core | shorthands⟩
```

## 12.17 Encoding issues (part 2)

`\TeX` `\LaTeX` Because documents may use font encodings other than one of the latin encodings, we make sure that the logos of TeX and LaTeX always come out in the right encoding.

```
⟨*core⟩
\bbl@redefine\TeX{\textlatin{\org@TeX}}
\bbl@redefine\LaTeX{\textlatin{\org@LaTeX}}
⟨/core⟩
```

## 12.18 Preventing clashes with other packages

### 12.18.1 `ifthen`

`\ifthenelse` Sometimes a document writer wants to create a special effect depending on the page a certain fragment of text appears on. This can be achieved by the following piece of code:

```
\ifthenelse{\isodd{\pageref{some:label}}}
          {code for odd pages}
          {code for even pages}
```

In order for this to work the argument of `\isodd` needs to be fully expandable. With the above redefinition of `\pageref` it is not in the case of this example. To overcome that, we add some code to the definition of `\ifthenelse` to make things work.

The first thing we need to do is check if the package `ifthen` is loaded. This should be done at `\begin{document}` time.

```
⟨*package⟩
\AtBeginDocument{%
  \@ifpackageloaded{ifthen}{%
```

Then we can redefine `\ifthenelse`:

```
    \bbl@redefine@long\ifthenelse#1#2#3{%
```

We want to revert the definition of `\pageref` to its original definition for the duration of `\ifthenelse`, so we first need to store its current meaning.

```
      \let\bbl@tempa\pageref
      \let\pageref\org@pageref
```

Then we can set the `\@safe@actives` switch and call the original `\ifthenelse`. In order to be able to use shorthands in the second and third arguments of `\ifthenelse` the resetting of the switch *and* the definition of `\pageref` happens inside those arguments.

```
      \@safe@activestrue
      \org@ifthenelse{#1}{%
        \let\pageref\bbl@tempa
        \@safe@activesfalse
        #2}{%
        \let\pageref\bbl@tempa
        \@safe@activesfalse
        #3}%
      }%
```

When the package wasn't loaded we do nothing.

```
    }{}%
  }
```

### 12.18.2 varioref

`\@@vpageref` `\vrefpagenum` `\Ref` When the package varioref is in use we need to modify its internal command `\@@vpageref` in order to prevent problems when an active character ends up in the argument of `\vref`.

```
\AtBeginDocument{%
  \@ifpackageloaded{varioref}{%
    \bbl@redefine\@@vpageref#1[#2]#3{%
      \@safe@activestrue
      \org@@@vpageref{#1}[#2]{#3}%
      \@safe@activesfalse}%
```

The same needs to happen for `\vrefpagenum`.

```
    \bbl@redefine\vrefpagenum#1#2{%
      \@safe@activestrue
      \org@vrefpagenum{#1}{#2}%
      \@safe@activesfalse}%
```

The package `varioref` defines `\Ref` to be a robust command wich uppercases the first character of the reference text. In order to be able to do that it needs to access the exandable form of `\ref`. So we employ a little trick here. We redefine the (internal) command `\Ref␣` to call `\org@ref` instead of `\ref`. The disadvantgage of this solution is that whenever the derfinition of `\Ref` changes, this definition needs to be updated as well.

```
    \expandafter\def\csname Ref \endcsname#1{%
      \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
    }{}%
  }
```

### 12.18.3 hhline

`\hhline` Delaying the activation of the shorthand characters has introduced a problem with the `hhline` package. The reason is that it uses the ':' character which is made active by the french support in babel. Therefor we need to *reload* the package when the ':' is an active character.

So at `\begin{document}` we check whether `hhline` is loaded.

```
\AtBeginDocument{%
  \@ifpackageloaded{hhline}%
```

Then we check whether the expansion of `\normal@char:` is not equal to `\relax`.

```
    {\expandafter\ifx\csname normal@char\string:\endcsname\relax
     \else
```

In that case we simply reload the package. Note that this happens *after* the category code of the @-sign has been changed to other, so we need to temporarily change it to letter again.

```
       \makeatletter
       \def\@currname{hhline}\input{hhline.sty}\makeatother
     \fi}%
    {}}
```

### 12.18.4 General

`\FOREIGNLANGUAGE` The package `fancyhdr` treats the running head and fout lines somewhat differently as the standard classes. A symptom of this is that the command `\foreignlanguage` which babel adds to the marks can end up inside the argument of `\MakeUppercase`. To prevent unexpected results we need to define `\FOREIGNLANGUAGE` here.

```
\DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
  \lowercase{\foreignlanguage{#1}}}
⟨/package⟩
```

`\nfss@catcodes` LaTeX's font selection scheme sometimes wants to read font definition files in the middle of processing the document. In order to guard against any characters having the wrong `\catcode`s it always calls `\nfss@catcodes` before loading a file. Unfortunately, the characters `"` and `'` are not dealt with. Therefor we have to add them until LaTeX does that herself.

```
⟨*core | shorthands⟩
\ifx\nfss@catcodes\@undefined
\else
  \addto\nfss@catcodes{%
    \@makeother\'%
    \@makeother\"%
    }
\fi

⟨/core | shorthands⟩
```

## 13 Local Language Configuration

`\loadlocalcfg` At some sites it may be necessary to add site-specific actions to a language definition file. This can be done by creating a file with the same name as the language definition file, but with the extension `.cfg`. For instance the file `norsk.cfg` will be loaded when the language definition file `norsk.ldf` is loaded.

```
⟨*core⟩
```

For plain-based formats we don't want to override the definition of `\loadlocalcfg` from `plain.def`.

```
\ifx\loadlocalcfg\@undefined
  \def\loadlocalcfg#1{%
    \InputIfFileExists{#1.cfg}
           {\typeout{*************************************^^J%
                          * Local config file #1.cfg used^^J%
                          *}%
            }
           {}}
\fi
```

Just to be compatible with LaTeX 2.09 we add a few more lines of code:

```
\ifx\@unexpandable@protect\@undefined
  \def\@unexpandable@protect{\noexpand\protect\noexpand}
  \long\def \protected@write#1#2#3{%
        \begingroup
         \let\thepage\relax
         #2%
         \let\protect\@unexpandable@protect
```

```
          \edef\reserved@a{\write#1{#3}}%
          \reserved@a
         \endgroup
         \if@nobreak\ifvmode\nobreak\fi\fi
  }
\fi
⟨/core⟩
```

## 14 Driver files for the documented source code

Since babel version 3.4 all source files that are part of the babel system can be typeset separately. But to typeset them all in one document, the file `babel.drv` can be used. If you only want the information on how to use the babel system and what goodies are provided by the language-specific files, you can run the file `user.drv` through LaTeX to get a user guide.

```
⟨*driver⟩
\documentclass{ltxdoc}
\usepackage{url,t1enc,supertabular}
\usepackage[icelandic,english]{babel}
\DoNotIndex{\!,\',\,,\.,\-,\:,\;,\?,\/,\^,\`,\@M}
\DoNotIndex{\@,\@ne,\@m,\@afterheading,\@date,\@endpart}
\DoNotIndex{\@hangfrom,\@idxitem,\@makeschapterhead,\@mkboth}
\DoNotIndex{\@oddfoot,\@oddhead,\@restonecolfalse,\@restonecoltrue}
\DoNotIndex{\@starttoc,\@unused}
\DoNotIndex{\accent,\active}
\DoNotIndex{\addcontentsline,\advance,\Alph,\arabic}
\DoNotIndex{\baselineskip,\begin,\begingroup,\bf,\box,\c@secnumdepth}
\DoNotIndex{\catcode,\centering,\char,\chardef,\clubpenalty}
\DoNotIndex{\columnsep,\columnseprule,\crcr,\csname}
\DoNotIndex{\day,\def,\dimen,\discretionary,\divide,\dp,\do}
\DoNotIndex{\edef,\else,\@empty,\end,\endgroup,\endcsname,\endinput}
\DoNotIndex{\errhelp,\errmessage,\expandafter,\fi,\filedate}
\DoNotIndex{\fileversion,\fmtname,\fnum@figure,\fnum@table,\fontdimen}
\DoNotIndex{\gdef,\global}
\DoNotIndex{\hbox,\hidewidth,\hfil,\hskip,\hspace,\ht,\Huge,\huge}
\DoNotIndex{\ialign,\if@twocolumn,\ifcase,\ifcat,\ifhmode,\ifmmode}
\DoNotIndex{\ifnum,\ifx,\immediate,\ignorespaces,\input,\item}
\DoNotIndex{\kern}
\DoNotIndex{\labelsep,\Large,\large,\labelwidth,\lccode,\leftmargin}
\DoNotIndex{\lineskip,\leavevmode,\let,\list,\ll,\long,\lower}
\DoNotIndex{\m@ne,\mathchar,\mathaccent,\markboth,\month,\multiply}
\DoNotIndex{\newblock,\newbox,\newcount,\newdimen,\newif,\newwrite}
\DoNotIndex{\nobreak,\noexpand,\noindent,\null,\number}
\DoNotIndex{\onecolumn,\or}
\DoNotIndex{\p@,par, \parbox,\parindent,\parskip,\penalty}
\DoNotIndex{\protect,\ps@headings}
\DoNotIndex{\quotation}
\DoNotIndex{\raggedright,\raise,\refstepcounter,\relax,\rm,\setbox}
\DoNotIndex{\section,\setcounter,\settowidth,\scriptscriptstyle}
\DoNotIndex{\sfcode,\sl,\sloppy,\small,\space,\spacefactor,\strut}
\DoNotIndex{\string}
\DoNotIndex{\textwidth,\the,\thechapter,\thefigure,\thepage,\thepart}
\DoNotIndex{\thetable,\thispagestyle,\titlepage,\tracingmacros}
\DoNotIndex{\tw@,\twocolumn,\typeout,\uppercase,\usecounter}
\DoNotIndex{\vbox,\vfil,\vskip,\vspace,\vss}
\DoNotIndex{\widowpenalty,\write,\xdef,\year,\z@,\z@skip}
```

Here `\dlqq` is defined so that an example of `"’` can be given.

```
\makeatletter
\gdef\dlqq{{\setbox\tw@=\hbox{,}\setbox\z@=\hbox{''}%
  \dimen\z@=\ht\z@ \advance\dimen\z@-\ht\tw@
  \setbox\z@=\hbox{\lower\dimen\z@\box\z@}\ht\z@=\ht\tw@
  \dp\z@=\dp\tw@ \box\z@\kern-.04em}}
```

The code lines are numbered within sections,

```
⟨*!user⟩
\@addtoreset{CodelineNo}{section}
\renewcommand\theCodelineNo{%
  \reset@font\scriptsize\thesection.\arabic{CodelineNo}}
```

which should also be visible in the index; hence this redefinition of a macro from `doc.sty`.

```
\renewcommand\codeline@wrindex[1]{\if@filesw
        \immediate\write\@indexfile
            {\string\indexentry{#1}%
            {\number\c@section.\number\c@CodelineNo}}\fi}
```

The glossary environment is used or the change log, but its definition needs changing for this document.

```
\renewenvironment{theglossary}{%
    \glossary@prologue%
    \GlossaryParms \let\item\@idxitem \ignorespaces}%
   {}
⟨/!user⟩
\makeatother
```

A few shorthands used in the documentation

```
\font\manual=logo10 % font used for the METAFONT logo, etc.
\newcommand*\MF{{\manual META}\-{\manual FONT}}
\newcommand*\TeXhax{\TeX hax}
\newcommand*\babel{\textsf{babel}}
\newcommand*\Babel{\textsf{Babel}}
\newcommand*\m[1]{\mbox{$\langle$\it#1\/$\rangle$}}
\newcommand*\langvar{\m{lang}}
```

Some more definitions needed in the documentation.

```
%\newcommand*\note[1]{\textbf{#1}}
\newcommand*\note[1]{}
\newcommand*\bsl{\protect\bslash}
\newcommand*\Lopt[1]{\textsf{#1}}
\newcommand*\Lenv[1]{\textsf{#1}}
\newcommand*\file[1]{\texttt{#1}}
\newcommand*\cls[1]{\texttt{#1}}
\newcommand*\pkg[1]{\texttt{#1}}
\newcommand*\langdeffile[1]{%
⟨−user⟩  \clearpage
  \DocInput{#1}}
```

When a full index should be generated uncomment the line with `\EnableCrossrefs`. Beware, processing may take some time. Use `\DisableCrossrefs` when the index is ready.

```
%  \EnableCrossrefs
\DisableCrossrefs
```

Inlude the change log.

```
⟨−user⟩\RecordChanges
```

The index should use the linenumbers of the code.

```
⟨−user⟩\CodelineIndex
```

Set everything in `\MacroFont` instead of `\AltMacroFont`

```
\setcounter{StandardModuleDepth}{1}
```

For the user guide we only want the description parts of all the files.

```
⟨+user⟩\OnlyDescription
```

Here starts the document

```
\begin{document}
\DocInput{babel.dtx}
```

All the language definition files.

```
⟨+user⟩\clearpage
\langdeffile{esperanto.dtx}
\langdeffile{interlingua.dtx}
%
\langdeffile{dutch.dtx}
\langdeffile{english.dtx}
\langdeffile{germanb.dtx}
\langdeffile{ngermanb.dtx}
%
\langdeffile{breton.dtx}
\langdeffile{welsh.dtx}
\langdeffile{irish.dtx}
\langdeffile{scottish.dtx}
%
\langdeffile{greek.dtx}
%
\langdeffile{frenchb.dtx}
\langdeffile{italian.dtx}
\langdeffile{latin.dtx}
\langdeffile{portuges.dtx}
\langdeffile{spanish.dtx}
\langdeffile{catalan.dtx}
\langdeffile{galician.dtx}
\langdeffile{basque.dtx}
\langdeffile{romanian.dtx}
%
\langdeffile{danish.dtx}
\langdeffile{icelandic.dtx}
\langdeffile{norsk.dtx}
```

```
\langdeffile{swedish.dtx}
\langdeffile{samin.dtx}
%
\langdeffile{finnish.dtx}
\langdeffile{magyar.dtx}
\langdeffile{estonian.dtx}
%
\langdeffile{croatian.dtx}
\langdeffile{czech.dtx}
\langdeffile{polish.dtx}
\langdeffile{serbian.dtx}
\langdeffile{slovak.dtx}
\langdeffile{slovene.dtx}
\langdeffile{russianb.dtx}
\langdeffile{bulgarian.dtx}
\langdeffile{ukraineb.dtx}
%
\langdeffile{lsorbian.dtx}
\langdeffile{usorbian.dtx}
\langdeffile{turkish.dtx}
%
\langdeffile{hebrew.dtx}
\DocInput{hebinp.dtx}
\DocInput{hebrew.fdd}
\DocInput{heb209.dtx}
\langdeffile{bahasa.dtx}
%\langdeffile{sanskrit.dtx}
%\langdeffile{kannada.dtx}
%\langdeffile{nagari.dtx}
%\langdeffile{tamil.dtx}
\clearpage
\DocInput{bbplain.dtx}
```

Finally print the index and change log (not for the user guide).

```
⟨*!user⟩
\clearpage
\def\filename{index}
\PrintIndex
\clearpage
\def\filename{changes}
\PrintChanges
⟨/!user⟩
\end{document}
⟨/driver⟩
```

## 15 Conclusion

A system of document options has been presented that enable the user of LaTeX to adapt the standard document classes of LaTeX to the language he or she prefers to use. These options offer the possibility of switching between languages in one document. The basic interface consists of using one option, which is the same for *all* standard document classes.

In some cases the language definition files provide macros that can be useful to plain TeX users as well as to LaTeX users. The babel system has been implemented so that it can be used by both groups of users.

## 16 Acknowledgements

I would like to thank all who volunteered as $\beta$-testers for their time. I would like to mention Julio Sanchez who supplied the option file for the Spanish language and Maurizio Codogno who supplied the option file for the Italian language. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped find and repair bugs.

During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful.

## References

[1] Donald E. Knuth, *The TeXbook*, Addison-Wesley, 1986.

[2] Leslie Lamport, *LaTeX, A document preparation System*, Addison-Wesley, 1986.

[3] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst.* SDU Uitgeverij ('s-Gravenhage, 1988). A Dutch book on layout design and typography.

[4] Hubert Partl, *German TeX*, *TUGboat* 9 (1988) #1, p. 70–72.

[5] Leslie Lamport, in: TeXhax Digest, Volume 89, #13, 17 February 1989.

[6] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national LaTeX styles*, *TUGboat* 10 (1989) #3, p. 401–406.

[7] Joachim Schrod, *International LaTeX is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.

## 17 The Esperanto language

The file `esperanto.dtx`[10] defines all the language-specific macros for the Esperanto language.

For this language the character `^` is made active. In table 2 an overview is given of its purpose.

| | |
|---|---|
| `^c` | gives ĉ with hyphenation in the rest of the word allowed, this works for c, C, g, G, H, J, s, S, z, Z |
| `^h` | prevents ĥ from becoming too tall |
| `^j` | gives ĵ |
| `^u` | gives ŭ, with hyphenation in the rest of the word allowed |
| `^U` | gives Ŭ, with hyphenation in the rest of the word allowed |
| `^|` | inserts a `\discretionary{-}{}{}` |

Table 2: The functions of the active character for Esperanto.

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
⟨*code⟩
\LdfInit{esperanto}\captionsesperanto
```

When this file is read as an option, i.e. by the `\usepackage` command, `esperanto` will be an 'unknown' language in which case we have to make it known. So we check for the existence of `\l@esperanto` to see whether we have to do something here.

```
\ifx\l@esperanto\@undefined
  \@nopatterns{Esperanto}
  \adddialect\l@esperanto0\fi
```

The next step consists of defining commands to switch to the Esperanto language. The reason for this is that a user might want to switch back and forth between languages.

`\captionsesperanto` The macro `\captionsesperanto` defines all strings used in the four standard documentclasses provided with LaTeX.

```
\addto\captionsesperanto{%
  \def\prefacename{Anta\u{u}parolo}%
  \def\refname{Cita\^\j{}oj}%
  \def\abstractname{Resumo}%
  \def\bibname{Bibliografio}%
  \def\chaptername{{\^C}apitro}%
  \def\appendixname{Apendico}%
```

[10]The file described in this section has version number ? and was last revised on ?. A contribution was made by Ruiz-Altaba Marti (`ruizaltb@cernvm.cern.ch`). Code from the file `esperant.sty` by Jörg Knappen (`knappen@vkpmzd.kph.uni-mainz.de`) was included.

```
  \def\contentsname{Enhavo}%
  \def\listfigurename{Listo de figuroj}%
  \def\listtablename{Listo de tabeloj}%
  \def\indexname{Indekso}%
  \def\figurename{Figuro}%
  \def\tablename{Tabelo}%
  \def\partname{Parto}%
  \def\enclname{Aldono(j)}%
  \def\ccname{Kopie al}%
  \def\headtoname{Al}%
  \def\pagename{Pa\^go}%
  \def\subjectname{Temo}%
  \def\seename{vidu}%   a^u: vd.
  \def\alsoname{vidu anka\u{u}}% a^u vd. anka\u{u}
  \def\proofname{Pruvo}%
  \def\glossaryname{Glosaro}%
  }
```

`\dateesperanto` The macro `\dateesperanto` redefines the command `\today` to produce Esperanto dates.

```
\def\dateesperanto{%
  \def\today{\number\day{--a}~de~\ifcase\month\or
    januaro\or februaro\or marto\or aprilo\or majo\or junio\or
    julio\or a\u{u}gusto\or septembro\or oktobro\or novembro\or
    decembro\fi,\space \number\year}}
```

`\extrasesperanto`
`\noextrasesperanto` The macro `\extrasesperanto` performs all the extra definitions needed for the Esperanto language. The macro `\noextrasesperanto` is used to cancel the actions of `\extrasesperanto`.

For Esperanto the `^` character is made active. This is done once, later on its definition may vary.

```
\initiate@active@char{^}
```

Because the character `^` is used in math mode with quite a different purpose we need to add an extra level of evaluation to the definition of the active `^`. It checks whether math mode is active; if so the shorthand mechanism is bypassed by a direct call of `\normal@char^`.

```
\addto\extrasesperanto{\languageshorthands{esperanto}}
\addto\extrasesperanto{\bbl@activate{^}}
\addto\noextrasesperanto{\bbl@deactivate{^}}
```

In order to prevent problems with the active `^` we add a shorthand on system level which expands to a 'normal `^`.

```
\declare@shorthand{system}{^}{\csname normal@char\string^\endcsname}
```

And here are the uses of the active `^`:

```
\declare@shorthand{esperanto}{^c}{\^{c}\allowhyphens}
\declare@shorthand{esperanto}{^C}{\^{C}\allowhyphens}
\declare@shorthand{esperanto}{^g}{\^{g}\allowhyphens}
```

```
\declare@shorthand{esperanto}{^G}{\^{G}\allowhyphens}
\declare@shorthand{esperanto}{^h}{h\llap{\^{}}\allowhyphens}
\declare@shorthand{esperanto}{^H}{\^{H}\allowhyphens}
\declare@shorthand{esperanto}{^j}{\^{\j}\allowhyphens}
\declare@shorthand{esperanto}{^J}{\^{J}\allowhyphens}
\declare@shorthand{esperanto}{^s}{\^{s}\allowhyphens}
\declare@shorthand{esperanto}{^S}{\^{S}\allowhyphens}
\declare@shorthand{esperanto}{^u}{\u u\allowhyphens}
\declare@shorthand{esperanto}{^U}{\u U\allowhyphens}
\declare@shorthand{esperanto}{^|}{\discretionary{-}{}{}\allowhyphens}
```

`\Esper` `\esper`

In `esperant.sty` Jörg Knappen provides the macros `\esper` and `\Esper` that can be used instead of `\alph` and `\Alph`. These macros are available in this file as well.

Their definition takes place in two steps. First the toplevel.

```
\def\esper#1{\@esper{\@nameuse{c@#1}}}
\def\Esper#1{\@Esper{\@nameuse{c@#1}}}
```

Then the second level.

```
\def\@esper#1{%
  \ifcase#1\or a\or b\or c\or \^c\or d\or e\or f\or g\or \^g\or
    h\or h\llap{\^{}}\or i\or j\or \^\j\or k\or l\or m\or n\or o\or
    p\or s\or \^s\or t\or u\or \u{u}\or v\or z\else\@ctrerr\fi}
\def\@Esper#1{%
  \ifcase#1\or A\or B\or C\or \^C\or D\or E\or F\or G\or \^G\or
    H\or \^H\or I\or J\or \^J\or K\or L\or M\or N\or O\or
    P\or S\or \^S\or T\or U\or \u{U}\or V\or Z\else\@ctrerr\fi}
```

`\hodiau` `\hodiaun`

In `esperant.sty` Jörg Knappen provides two alternative macros for `\today`, `\hodiau` and `\hodiaun`. The second macro produces an accusative version of the date in Esperanto.

```
\addto\dateesperanto{\def\hodiau{la \today}}
\def\hodiaun{la \number\day --an~de~\ifcase\month\or
  januaro\or februaro\or marto\or aprilo\or majo\or junio\or
  julio\or a\u{u}gusto\or septembro\or oktobro\or novembro\or
  decembro\fi, \space \number\year}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{esperanto}
⟨/code⟩
```

## 18 The Interlingua language

The file `interlingua.dtx`[11] defines all the language definition macros for the Interlingua language. This file was contributed by Peter Kleiweg, kleiweg at let.rug.nl.

Interlingua is an auxiliary language, built from the common vocabulary of Spanish/Portuguese, English, Italian and French, with some normalisation of spelling. The grammar is very easy, more similar to English's than to neolatin languages. The site `http://www.interlingua.com` is mostly written in interlingua (as is `http://interlingua.altervista.org`), in case you want to read some sample of it.

You can have a look at the grammar at `http://www.geocities.com/linguablau`

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
⟨*code⟩
\LdfInit{interlingua}{captionsinterlingua}
```

When this file is read as an option, i.e. by the `\usepackage` command, `interlingua` could be an 'unknown' language in which case we have to make it known. So we check for the existence of `\l@interlingua` to see whether we have to do something here.

```
\ifx\undefined\l@interlingua
  \@nopatterns{Interlingua}
  \adddialect\l@interlingua0\fi
```

The next step consists of defining commands to switch to (and from) the Interlingua language.

`\interlinguahyphenmins` This macro is used to store the correct values of the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`.

```
\providehyphenmins{interlingua}{\tw@\tw@}
```

`\captionsinterlingua` The macro `\captionsinterlingua` defines all strings used in the four standard documentclasses provided with LaTeX.

```
\def\captionsinterlingua{%
  \def\prefacename{Prefacio}%
  \def\refname{Referentias}%
  \def\abstractname{Summario}%
  \def\bibname{Bibliographia}%
  \def\chaptername{Capitulo}%
  \def\appendixname{Appendice}%
  \def\contentsname{Contento}%
  \def\listfigurename{Lista de figuras}%
  \def\listtablename{Lista de tabellas}%
  \def\indexname{Indice}%
  \def\figurename{Figura}%
  \def\tablename{Tabella}%
```

[11]The file described in this section has version number v1.6 and was last revised on 2005/03/30.

```
  \def\partname{Parte}%
  \def\enclname{Incluso}%
  \def\ccname{Copia}%
  \def\headtoname{A}%
  \def\pagename{Pagina}%
  \def\seename{vide}%
  \def\alsoname{vide etiam}%
  \def\proofname{Prova}%
  \def\glossaryname{Glossario}%
  }
```

`\dateinterlingua` The macro `\dateinterlingua` redefines the command `\today` to produce Interlingua dates.

```
\def\dateinterlingua{%
  \def\today{le~\number\day\space de \ifcase\month\or
    januario\or februario\or martio\or april\or maio\or junio\or
    julio\or augusto\or septembre\or octobre\or novembre\or
    decembre\fi
    \space \number\year}}
```

`\extrasinterlingua` `\noextrasinterlingua` The macro `\extrasinterlingua` will perform all the extra definitions needed for the Interlingua language. The macro `\noextrasinterlingua` is used to cancel the actions of `\extrasinterlingua`. For the moment these macros are empty but they are defined for compatibility with the other language definition files.

```
\addto\extrasinterlingua{}
\addto\noextrasinterlingua{}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{interlingua}
⟨/code⟩
```

## 19 The Dutch language

The file `dutch.dtx`[12] defines all the language-specific macros for the Dutch language and the 'Afrikaans' version[13] of it.

For this language the character `"` is made active. In table 3 an overview is given of its purpose. One of the reasons for this is that in the Dutch language a word with a dieresis can be hyphenated just before the letter with the umlaut, but the dieresis has to disappear if the word is broken between the previous letter and the accented letter.

In [3] the quoting conventions for the Dutch language are discussed. The preferred convention is the single-quote Anglo-American convention, i.e. 'This is a quote'. An alternative is the slightly old-fashioned Dutch method with initial double quotes lowered to the baseline, „This is a quote", which should be typed as `"‘This is a quote"’`.

| | |
|---|---|
| `"a` | `\"a` which hyphenates as `-a`; also implemented for the other letters. |
| `"y` | puts a negative kern between `i` and `j` |
| `"Y` | puts a negative kern between `I` and `J` |
| `"\|` | disable ligature at this position. |
| `"-` | an explicit hyphen sign, allowing hyphenation in the rest of the word. |
| `"~` | to produce a hyphencharcter without the following `\discretionary{}{}{}`. |
| `""` | to produce an invisible 'breakpoint'. |
| `"‘` | lowered double left quotes (see example below). |
| `"’` | normal double right quotes. |
| `\-` | like the old `\-`, but allowing hyphenation in the rest of the word. |

Table 3: The extra definitions made by `dutch.ldf`

```
% \changes{dutch-3.8a}{1996/10/04}{made check dependant on
%    \cs{CurrentOption}}
%
%    The macro |\LdfInit| takes care of preventing that this file is
%    loaded more than once, checking the category code of the
%    \texttt{@} sign, etc.
% \changes{dutch-3.8a}{1996/10/30}{Now use \cs{LdfInit} to perform
%    initial checks}
%    \begin{macrocode}
⟨*code⟩
\LdfInit\CurrentOption{captions\CurrentOption}
```

[12]The file described in this section has version number v3.8i, and was last revised on 2005/03/30.

[13]contributed by Stoffel Lombard (`lombc@b31pc87.up.ac.za`)

When this file is read as an option, i.e. by the `\usepackage` command, `dutch` could be an 'unknown' language in which case we have to make it known. So we check for the existence of `\l@dutch` or `l@afrikaans` to see whether we have to do something here.

First we try to establish with which option we are being processed.

```
\def\bbl@tempa{dutch}
\ifx\CurrentOption\bbl@tempa
```

If it is `dutch` then we first check if the Dutch hyphenation patterns wer loaded,

```
  \ifx\l@dutch\undefined
```

if no we issue a warning and make dutch a 'dialect' of either the hyphenation patterns that were loaded in slot 0 or of 'afrikaans' when it is available.

```
    \@nopatterns{Dutch}
    \ifx\l@afrikaans\undefined
      \adddialect\l@dutch0
    \else
      \adddialect\l@dutch\l@afrikaans
    \fi
  \fi
```

The next step consists of defining commands to switch to (and from) the Dutch language.

`\captionsdutch` The macro `\captionsdutch` defines all strings used in the four standard document classes provided with LaTeX.

```
  \begingroup
    \catcode`\"\active
    \def\x{\endgroup
      \def\captionsdutch{%
        \def\prefacename{Voorwoord}%
        \def\refname{Referenties}%
        \def\abstractname{Samenvatting}%
        \def\bibname{Bibliografie}%
        \def\chaptername{Hoofdstuk}%
        \def\appendixname{B"ylage}%
        \def\contentsname{Inhoudsopgave}%
        \def\listfigurename{L"yst van figuren}%
        \def\listtablename{L"yst van tabellen}%
        \def\indexname{Index}%
        \def\figurename{Figuur}%
        \def\tablename{Tabel}%
        \def\partname{Deel}%
        \def\enclname{B"ylage(n)}%
        \def\ccname{cc}%
        \def\headtoname{Aan}%
        \def\pagename{Pagina}%
        \def\seename{zie}%
        \def\alsoname{zie ook}%
        \def\proofname{Bew"ys}%
      \def\glossaryname{Verklarende Woordenl"yst}%
      }
    }\x
```

\datedutch The macro `\datedutch` redefines the command `\today` to produce Dutch dates.

```
  \def\datedutch{%
    \def\today{\number\day~\ifcase\month\or
      januari\or februari\or maart\or april\or mei\or juni\or
      juli\or augustus\or september\or oktober\or november\or
      december\fi
      \space \number\year}}
```

When the option with which this file is being process was not dutch we assume it was afrikaans. We perform a similar check on the availability of the hyphenation paterns.

```
\else
  \ifx\l@afrikaans\undefined
    \@nopatterns{Afrikaans}
    \ifx\l@dutch\undefined
      \adddialect\l@afrikaans0
    \else
      \adddialect\l@afrikaans\l@dutch
    \fi
  \fi
```

\captionsafrikaans Now is the time to define the words for ‘Afrikaans’.

```
  \def\captionsafrikaans{%
    \def\prefacename{Voorwoord}%
    \def\refname{Verwysings}%
    \def\abstractname{Samevatting}%
    \def\bibname{Bibliografie}%
    \def\chaptername{Hoofstuk}%
    \def\appendixname{Bylae}%
    \def\contentsname{Inhoudsopgawe}%
    \def\listfigurename{Lys van figure}%
    \def\listtablename{Lys van tabelle}%
    \def\indexname{Inhoud}%
    \def\figurename{Figuur}%
    \def\tablename{Tabel}%
    \def\partname{Deel}%
    \def\enclname{Bylae(n)}%
    \def\ccname{a.a.}%
    \def\headtoname{Aan}%
    \def\pagename{Bladsy}%
    \def\seename{sien}%
    \def\alsoname{sien ook}%
    \def\proofname{Bewys}%
    }
```

`\dateafrikaans` Here is the ‘Afrikaans’ version of the date macro.

```
 \def\dateafrikaans{%
   \def\today{\number\day~\ifcase\month\or
     Januarie\or Februarie\or Maart\or April\or Mei\or Junie\or
     Julie\or  Augustus\or September\or Oktober\or November\or
     Desember\fi
     \space \number\year}}
\fi
```

`\extrasdutch` `\extrasafrikaans` `\noextrasdutch` `\noextrasafrikaans` The macros `\extrasdutch` and `\captionsafrikaans` will perform all the extra definitions needed for the Dutch language. The macros `\noextrasdutch` and `noextrasafrikaans` is used to cancel the actions of `\extrasdutch` and `\captionsafrikaans`.

For Dutch the `"` character is made active. This is done once, later on its definition may vary. Other languages in the same document may also use the `"` character for shorthands; we specify that the dutch group of shorthands should be used.

```
\initiate@active@char{"}
```

Both version of the language use the same set of shorthand definitions althoug the ‘ij’ is not used in Afrikaans.

```
\@namedef{extras\CurrentOption}{\languageshorthands{dutch}}
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \bbl@activate{"}}
```

The ‘umlaut’ character should be positioned lower on *all* vowels in Dutch texts.

```
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \umlautlow\umlautelow}
\@namedef{noextras\CurrentOption}{%
  \umlauthigh}
```

`\dutchhyphenmins` `\afrikaanshyphenmins` The dutch hyphenation patterns can be used with `\lefthyphenmin` set to 2 and `\righthyphenmin` set to 3.

```
\providehyphenmins{\CurrentOption}{\tw@\thr@@}
```

`\@trema` In the Dutch language vowels with a trema are treated specially. If a hyphenation occurs before a vowel-plus-trema, the trema should disappear. To be able to do this we could first define the hyphenation break behaviour for the five vowels, both lowercase and uppercase, in terms of `\discretionary`. But this results in a large `\if`-construct in the definition of the active `"`. Because we think a user should not use `"` when he really means something like `''` we chose not to distinguish between vowels and consonants. Therefore we have one macro `\@trema` which specifies the hyphenation break behaviour for all letters.

```
\def\@trema#1{\allowhyphens\discretionary{-}{#1}{\"{#1}}\allowhyphens}
```

Now we can define the doublequote macros: the tremas,

```
\declare@shorthand{dutch}{"a}{\textormath{\@trema a}{\ddot a}}
```

```
\declare@shorthand{dutch}{"e}{\textormath{\@trema e}{\ddot e}}
\declare@shorthand{dutch}{"i}{\textormath
  {\allowhyphens\discretionary{-}{i}{\"{\i}}\allowhyphens}%
  {\ddot \imath}}
\declare@shorthand{dutch}{"o}{\textormath{\@trema o}{\ddot o}}
\declare@shorthand{dutch}{"u}{\textormath{\@trema u}{\ddot u}}
```

dutch quotes,

```
\declare@shorthand{dutch}{"`}{%
  \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
\declare@shorthand{dutch}{"'}{%
  \textormath{\textquotedblright}{\mbox{\textquotedblright}}}
```

and some additional commands:

```
\declare@shorthand{dutch}{"-}{\nobreak-\bbl@allowhyphens}
\declare@shorthand{dutch}{"~}{\textormath{\leavevmode\hbox{-}}{-}}
\declare@shorthand{dutch}{"|}{%
  \textormath{\discretionary{-}{}{\kern.03em}}{}}
\declare@shorthand{dutch}{""}{\hskip\z@skip}
\declare@shorthand{dutch}{"y}{\textormath{\ij{}}{\ddot y}}
\declare@shorthand{dutch}{"Y}{\textormath{\IJ{}}{\ddot Y}}
```

To enable hyphenation in two words, written together but separated by a slash, as in ‘uitdrukking/opmerking’ we define the command `"/`.

```
\declare@shorthand{dutch}{"/}{\textormath
  {\bbl@allowhyphens\discretionary{/}{}{/}\bbl@allowhyphens}{}}
```

`\-` All that is left now is the redefinition of `\-`. The new version of `\-` should indicate an extra hyphenation position, while allowing other hyphenation positions to be generated automatically. The standard behaviour of TeX in this respect is very unfortunate for languages such as Dutch and German, where long compound words are quite normal and all one needs is a means to indicate an extra hyphenation position on top of the ones that TeX can generate from the hyphenation patterns.

```
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \babel@save\-}
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \def\-{\bbl@allowhyphens\discretionary{-}{}{}\bbl@allowhyphens}}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish\CurrentOption
⟨/code⟩
```

## 20 The English language

The file `english.dtx`[14] defines all the language definition macros for the English language as well as for the American and Australian version of this language. For the Australian version the British hyphenation patterns will be used, if available, for the Canadian variant the American patterns are selected.

For this language currently no special definitions are needed or available.

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
⟨*code⟩
\LdfInit\CurrentOption{date\CurrentOption}
```

When this file is read as an option, i.e. by the `\usepackage` command, `english` could be an ‘unknown’ language in which case we have to make it known. So we check for the existence of `\l@english` to see whether we have to do something here.

We allow for the british english patterns to be loaded as either ‘british’, or ‘UKenglish’. When neither of those is known we try to define `\l@english` as an alias for `\l@american` or `\l@USenglish`.

```
\ifx\l@english\@undefined
  \ifx\l@UKenglish\@undefined
    \ifx\l@british\@undefined
      \ifx\l@american\@undefined
        \ifx\l@USenglish\@undefined
          \ifx\l@canadian\@undefined
            \ifx\l@australian\@undefined
              \ifx\l@newzealand\@undefined
                \@nopatterns{English}
                \adddialect\l@english0
              \else
                \let\l@english\l@newzealand
              \fi
            \else
              \let\l@english\l@australian
            \fi
          \else
            \let\l@english\l@canadian
          \fi
        \else
          \let\l@english\l@USenglish
        \fi
      \else
        \let\l@english\l@american
      \fi
    \else
      \let\l@english\l@british
```

[14]The file described in this section has version number v3.3o and was last revised on 2005/03/30.

```
    \fi
  \else
    \let\l@english\l@UKenglish
  \fi
\fi
```

Because we allow 'british' to be used as the babel option we need to make sure that it will be recognised by `\selectlanguage`. In the code above we have made sure that `\l@english` was defined. Now we want to make sure that `\l@british` and `\l@UKenglish` are defined as well. When either of them is we make them equal to each other, when neither is we fall back to the default, `\l@english`.

```
\ifx\l@british\@undefined
  \ifx\l@UKenglish\@undefined
    \adddialect\l@british\l@english
    \adddialect\l@UKenglish\l@english
  \else
    \let\l@british\l@UKenglish
  \fi
\else
  \let\l@UKenglish\l@british
\fi
```

'American' is a version of 'English' which can have its own hyphenation patterns. The default english patterns are in fact for american english. We allow for the patterns to be loaded as 'english' 'american' or 'USenglish'.

```
\ifx\l@american\@undefined
  \ifx\l@USenglish\@undefined
```

When the patterns are not know as 'american' or 'USenglish' we add a "dialect".

```
    \adddialect\l@american\l@english
  \else
    \let\l@american\l@USenglish
  \fi
\else
```

Make sure that USenglish is known, even if the patterns were loaded as 'american'.

```
  \ifx\l@USenglish\@undefined
    \let\l@USenglish\l@american
  \fi
\fi
```

'Canadian' english spelling is a hybrid of British and American spelling. Although so far no special 'translations' have been reported we allow this file to be loaded by the option candian as well.

```
\ifx\l@canadian\@undefined
  \adddialect\l@canadian\l@american
\fi
```

'Australian' and 'New Zealand' english spelling seem to be the same as British spelling. Although so far no special 'translations' have been reported we allow this file to be loaded by the options australian and newzealand as well.

```
\ifx\l@australian\@undefined
  \adddialect\l@australian\l@british
\fi
\ifx\l@newzealand\@undefined
  \adddialect\l@newzealand\l@british
\fi
```

`\englishhyphenmins` This macro is used to store the correct values of the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`.

```
\providehyphenmins{\CurrentOption}{\tw@\thr@@}
```

The next step consists of defining commands to switch to (and from) the English language.

`\captionsenglish` The macro `\captionsenglish` defines all strings used in the four standard document classes provided with LaTeX.

```
\@namedef{captions\CurrentOption}{%
  \def\prefacename{Preface}%
  \def\refname{References}%
  \def\abstractname{Abstract}%
  \def\bibname{Bibliography}%
  \def\chaptername{Chapter}%
  \def\appendixname{Appendix}%
  \def\contentsname{Contents}%
  \def\listfigurename{List of Figures}%
  \def\listtablename{List of Tables}%
  \def\indexname{Index}%
  \def\figurename{Figure}%
  \def\tablename{Table}%
  \def\partname{Part}%
  \def\enclname{encl}%
  \def\ccname{cc}%
  \def\headtoname{To}%
  \def\pagename{Page}%
  \def\seename{see}%
  \def\alsoname{see also}%
  \def\proofname{Proof}%
  \def\glossaryname{Glossary}%
  }
```

`\dateenglish` In order to define `\today` correctly we need to know whether it should be 'english', 'australian', or 'american'. We can find this out by checking the value of `\CurrentOption`.

```
\def\bbl@tempa{british}
\ifx\CurrentOption\bbl@tempa\def\bbl@tempb{UK}\fi
\def\bbl@tempa{UKenglish}
\ifx\CurrentOption\bbl@tempa\def\bbl@tempb{UK}\fi
\def\bbl@tempa{american}
\ifx\CurrentOption\bbl@tempa\def\bbl@tempb{US}\fi
```

```
\def\bbl@tempa{USenglish}
\ifx\CurrentOption\bbl@tempa\def\bbl@tempb{US}\fi
\def\bbl@tempa{canadian}
\ifx\CurrentOption\bbl@tempa\def\bbl@tempb{US}\fi
\def\bbl@tempa{australian}
\ifx\CurrentOption\bbl@tempa\def\bbl@tempb{AU}\fi
\def\bbl@tempa{newzealand}
\ifx\CurrentOption\bbl@tempa\def\bbl@tempb{AU}\fi
```

The macro `\dateenglish` redefines the command `\today` to produce English dates.

```
\def\bbl@tempa{UK}
\ifx\bbl@tempa\bbl@tempb
  \@namedef{date\CurrentOption}{%
    \def\today{\ifcase\day\or
      1st\or 2nd\or 3rd\or 4th\or 5th\or
      6th\or 7th\or 8th\or 9th\or 10th\or
      11th\or 12th\or 13th\or 14th\or 15th\or
      16th\or 17th\or 18th\or 19th\or 20th\or
      21st\or 22nd\or 23rd\or 24th\or 25th\or
      26th\or 27th\or 28th\or 29th\or 30th\or
      31st\fi~\ifcase\month\or
      January\or February\or March\or April\or May\or June\or
      July\or August\or September\or October\or November\or
      December\fi\space \number\year}}
```

`\dateaustralian` Now, test for 'australian' or 'american'.

```
\else
```

The macro `\dateaustralian` redefines the command `\today` to produce Australian resp. New Zealand dates.

```
  \def\bbl@tempa{AU}
  \ifx\bbl@tempa\bbl@tempb
    \@namedef{date\CurrentOption}{%
      \def\today{\number\day~\ifcase\month\or
        January\or February\or March\or April\or May\or June\or
        July\or August\or September\or October\or November\or
        December\fi\space \number\year}}
```

`\dateamerican` The macro `\dateamerican` redefines the command `\today` to produce American dates.

```
  \else
    \@namedef{date\CurrentOption}{%
      \def\today{\ifcase\month\or
        January\or February\or March\or April\or May\or June\or
        July\or August\or September\or October\or November\or
        December\fi \space\number\day, \number\year}}
  \fi
\fi
```

`\extrasenglish` `\noextrasenglish` The macro `\extrasenglish` will perform all the extra definitions needed for the English language. The macro `\noextrasenglish` is used to cancel the actions of `\extrasenglish`. For the moment these macros are empty but they are defined for compatibility with the other language definition files.

```
\@namedef{extras\CurrentOption}{}
\@namedef{noextras\CurrentOption}{}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish\CurrentOption
⟨/code⟩
```

## 21 The German language

The file `germanb.dtx`[15] defines all the language definition macros for the German language as well as for the Austrian dialect of this language[16].

For this language the character `"` is made active. In table 4 an overview is given of its purpose. One of the reasons for this is that in the German language some character combinations change when a word is broken between the combination. Also the vertical placement of the umlaut can be controlled this way. The quotes

| | |
|---|---|
| `"a` | `\"a`, also implemented for the other lowercase and uppercase vowels. |
| `"s` | to produce the German ß (like `\ss{}`). |
| `"z` | to produce the German ß (like `\ss{}`). |
| `"ck` | for `ck` to be hyphenated as `k-k`. |
| `"ff` | for `ff` to be hyphenated as `ff-f`, this is also implemented for l, m, n, p, r and t |
| `"S` | for `SS` to be `\uppercase{"s}`. |
| `"Z` | for `SZ` to be `\uppercase{"z}`. |
| `"\|` | disable ligature at this position. |
| `"-` | an explicit hyphen sign, allowing hyphenation in the rest of the word. |
| `""` | like `"-`, but producing no hyphen sign (for compund words with hyphen, e.g. `x-""y`). |
| `"~` | for a compound word mark without a breakpoint. |
| `"=` | for a compound word mark with a breakpoint, allowing hyphenation in the composing words. |
| `"‘` | for German left double quotes (looks like „). |
| `"’` | for German right double quotes. |
| `"<` | for French left double quotes (similar to <<). |
| `">` | for French right double quotes (similar to >>). |

Table 4: The extra definitions made by `german.ldf`

in table 4 can also be typeset by using the commands in table 5.

When this file was read through the option germanb we make it behave as if german was specified.

```
\def\bbl@tempa{germanb}
\ifx\CurrentOption\bbl@tempa
  \def\CurrentOption{german}
\fi
```

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

[15]The file described in this section has version number v2.6k and was last revised on 2004/02/19.

[16]This file is a re-implementation of Hubert Partl's `german.sty` version 2.5b, see [4].

| | |
|---|---|
| `\glqq` | for German left double quotes (looks like „). |
| `\grqq` | for German right double quotes (looks like “). |
| `\glq` | for German left single quotes (looks like ‚). |
| `\grq` | for German right single quotes (looks like ‘). |
| `\flqq` | for French left double quotes (similar to <<). |
| `\frqq` | for French right double quotes (similar to >>). |
| `\flq` | for (French) left single quotes (similar to <). |
| `\frq` | for (French) right single quotes (similar to >). |
| `\dq` | the original quotes character ("). |

Table 5: More commands which produce quotes, defined by `german.ldf`

```
⟨*code⟩
\LdfInit\CurrentOption{captions\CurrentOption}
```

When this file is read as an option, i.e., by the `\usepackage` command, `german` will be an ‘unknown’ language, so we have to make it known. So we check for the existence of `\l@german` to see whether we have to do something here.

```
\ifx\l@german\@undefined
  \@nopatterns{German}
  \adddialect\l@german0
\fi
```

For the Austrian version of these definitions we just add another language.

```
\adddialect\l@austrian\l@german
```

The next step consists of defining commands to switch to (and from) the German language.

`\captionsgerman` `\captionsaustrian` Either the macro `\captionsgerman` or the macro `\captionsaustrian` will define all strings used in the four standard document classes provided with LaTeX.

```
\@namedef{captions\CurrentOption}{%
  \def\prefacename{Vorwort}%
  \def\refname{Literatur}%
  \def\abstractname{Zusammenfassung}%
  \def\bibname{Literaturverzeichnis}%
  \def\chaptername{Kapitel}%
  \def\appendixname{Anhang}%
  \def\contentsname{Inhaltsverzeichnis}%    % oder nur: Inhalt
  \def\listfigurename{Abbildungsverzeichnis}%
  \def\listtablename{Tabellenverzeichnis}%
  \def\indexname{Index}%
  \def\figurename{Abbildung}%
  \def\tablename{Tabelle}%                  % oder: Tafel
  \def\partname{Teil}%
  \def\enclname{Anlage(n)}%                 % oder: Beilage(n)
  \def\ccname{Verteiler}%                   % oder: Kopien an
  \def\headtoname{An}%
  \def\pagename{Seite}%
  \def\seename{siehe}%
  \def\alsoname{siehe auch}%
  \def\proofname{Beweis}%
  \def\glossaryname{Glossar}%
  }
```

`\dategerman` The macro `\dategerman` redefines the command `\today` to produce German dates.

```
\def\month@german{\ifcase\month\or
  Januar\or Februar\or M\"arz\or April\or Mai\or Juni\or
  Juli\or August\or September\or Oktober\or November\or Dezember\fi}
\def\dategerman{\def\today{\number\day.~\month@german
    \space\number\year}}
```

`\dateaustrian` The macro `\dateaustrian` redefines the command `\today` to produce Austrian version of the German dates.

```
\def\dateaustrian{\def\today{\number\day.~\ifnum1=\month
  J\"anner\else \month@german\fi \space\number\year}}
```

`\extrasgerman` `\extrasaustrian` `\noextrasgerman` `\noextrasaustrian` Either the macro `\extrasgerman` or the macros `\extrasaustrian` will perform all the extra definitions needed for the German language. The macro `\noextrasgerman` is used to cancel the actions of `\extrasgerman`.

For German (as well as for Dutch) the `"` character is made active. This is done once, later on its definition may vary.

```
\initiate@active@char{"}
\@namedef{extras\CurrentOption}{%
  \languageshorthands{german}}
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \bbl@activate{"}}
```

Don't forget to turn the shorthands off again.

```
\addto\noextrasgerman{\bbl@deactivate{"}}
```

In order for TeX to be able to hyphenate German words which contain 'ß' (in the `OT1` position `^^Y`) we have to give the character a nonzero `\lccode` (see Appendix H, the TeXbook).

```
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \babel@savevariable{\lccode25}%
  \lccode25=25}
```

The umlaut accent macro `\"` is changed to lower the umlaut dots. The redefinition is done with the help of `\umlautlow`.

```
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \babel@save\"\umlautlow}
\@namedef{noextras\CurrentOption}{\umlauthigh}
```

The german hyphenation patterns can be used with `\lefthyphenmin` and `\righthyphenmin` set to 2.

```
\providehyphenmins{\CurrentOption}{\tw@\tw@}
```

For German texts we need to make sure that `\frenchspacing` is turned on.

```
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \bbl@frenchspacing}
\expandafter\addto\csname noextras\CurrentOption\endcsname{%
  \bbl@nonfrenchspacing}
```

The code above is necessary because we need an extra active character. This character is then used as indicated in table 4.

To be able to define the function of `"`, we first define a couple of 'support' macros.

`\dq` We save the original double quote character in `\dq` to keep it available, the math accent `\"` can now be typed as `"`.

```
\begingroup \catcode`\"12
\def\x{\endgroup
  \def\@SS{\mathchar"7019 }
  \def\dq{"}}
\x
```

Now we can define the doublequote macros: the umlauts,

```
\declare@shorthand{german}{"a}{\textormath{\"{a}\allowhyphens}{\ddot a}}
\declare@shorthand{german}{"o}{\textormath{\"{o}\allowhyphens}{\ddot o}}
\declare@shorthand{german}{"u}{\textormath{\"{u}\allowhyphens}{\ddot u}}
\declare@shorthand{german}{"A}{\textormath{\"{A}\allowhyphens}{\ddot A}}
\declare@shorthand{german}{"O}{\textormath{\"{O}\allowhyphens}{\ddot O}}
\declare@shorthand{german}{"U}{\textormath{\"{U}\allowhyphens}{\ddot U}}
```

tremas,

```
\declare@shorthand{german}{"e}{\textormath{\"{e}}{\ddot e}}
\declare@shorthand{german}{"E}{\textormath{\"{E}}{\ddot E}}
\declare@shorthand{german}{"i}{\textormath{\"{\i}}%
                                {\ddot\imath}}
\declare@shorthand{german}{"I}{\textormath{\"{I}}{\ddot I}}
```

german es-zet (sharp s),

```
\declare@shorthand{german}{"s}{\textormath{\ss}{\@SS{}}}
\declare@shorthand{german}{"S}{\SS}
\declare@shorthand{german}{"z}{\textormath{\ss}{\@SS{}}}
\declare@shorthand{german}{"Z}{SZ}
```

german and french quotes,

```
\declare@shorthand{german}{"`}{\glqq}
\declare@shorthand{german}{"'}{\grqq}
\declare@shorthand{german}{"<}{\flqq}
\declare@shorthand{german}{">}{\frqq}
```

discretionary commands

```
\declare@shorthand{german}{"c}{\textormath{\bbl@disc ck}{c}}
\declare@shorthand{german}{"C}{\textormath{\bbl@disc CK}{C}}
\declare@shorthand{german}{"F}{\textormath{\bbl@disc F{FF}}{F}}
\declare@shorthand{german}{"l}{\textormath{\bbl@disc l{ll}}{l}}
```

```
\declare@shorthand{german}{"L}{\textormath{\bbl@disc L{LL}}{L}}
\declare@shorthand{german}{"m}{\textormath{\bbl@disc m{mm}}{m}}
\declare@shorthand{german}{"M}{\textormath{\bbl@disc M{MM}}{M}}
\declare@shorthand{german}{"n}{\textormath{\bbl@disc n{nn}}{n}}
\declare@shorthand{german}{"N}{\textormath{\bbl@disc N{NN}}{N}}
\declare@shorthand{german}{"p}{\textormath{\bbl@disc p{pp}}{p}}
\declare@shorthand{german}{"P}{\textormath{\bbl@disc P{PP}}{P}}
\declare@shorthand{german}{"r}{\textormath{\bbl@disc r{rr}}{r}}
\declare@shorthand{german}{"R}{\textormath{\bbl@disc R{RR}}{R}}
\declare@shorthand{german}{"t}{\textormath{\bbl@disc t{tt}}{t}}
\declare@shorthand{german}{"T}{\textormath{\bbl@disc T{TT}}{T}}
```

We need to treat `"f` a bit differently in order to preserve the ff-ligature.

```
\declare@shorthand{german}{"f}{\textormath{\bbl@discff}{f}}
\def\bbl@discff{\penalty\@M
  \afterassignment\bbl@insertff \let\bbl@nextff= }
\def\bbl@insertff{%
  \if f\bbl@nextff
    \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi
  {\relax\discretionary{ff-}{f}{ff}\allowhyphens}{f\bbl@nextff}}
\let\bbl@nextff=f
```

and some additional commands:

```
\declare@shorthand{german}{"-}{\nobreak\-\bbl@allowhyphens}
\declare@shorthand{german}{"|}{%
  \textormath{\penalty\@M\discretionary{-}{}{\kern.03em}%
              \allowhyphens}{}}
\declare@shorthand{german}{""}{\hskip\z@skip}
\declare@shorthand{german}{"~}{\textormath{\leavevmode\hbox{-}}{-}}
\declare@shorthand{german}{"=}{\penalty\@M-\hskip\z@skip}
```

`\mdqon` `\mdqoff` `\ck` All that's left to do now is to define a couple of commands for reasons of compatibility with `german.sty`.

```
\def\mdqon{\shorthandon{"}}
\def\mdqoff{\shorthandoff{"}}
\def\ck{\allowhyphens\discretionary{k-}{k}{ck}\allowhyphens}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish\CurrentOption
⟨/code⟩
```

## 22 The German language – new orthography

The file `ngermanb.dtx`[17] defines all the language definition macros for the German language with the 'new orthography' introduced in August 1998. This includes also the Austrian dialect of this language.

As with the 'traditional' German orthography, the character `"` is made active, and the commands in table 4 can be used, except for `"ck` and `"ff` etc., which are no longer required.

The internal language names are `ngerman` and `naustrian`.

When this file was read through the option ngermanb we make it behave as if ngerman was specified.

```
22.1 \def\bbl@tempa{ngermanb}
22.2 \ifx\CurrentOption\bbl@tempa
22.3   \def\CurrentOption{ngerman}
22.4 \fi
```

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
22.5 ⟨*code⟩
22.6 \LdfInit\CurrentOption{captions\CurrentOption}
```

When this file is read as an option, i.e., by the `\usepackage` command, `ngerman` will be an 'unknown' language, so we have to make it known. So we check for the existence of `\l@ngerman` to see whether we have to do something here.

```
22.7 \ifx\l@ngerman\@undefined
22.8   \@nopatterns{ngerman}
22.9   \adddialect\l@ngerman0
22.10 \fi
```

For the Austrian version of these definitions we just add another language.

```
22.11 \adddialect\l@naustrian\l@ngerman
```

The next step consists of defining commands to switch to (and from) the German language.

`\captionsngerman` `\captionsnaustrian` Either the macro `\captionnsgerman` or the macro `\captionsnaustrian` will define all strings used in the four standard document classes provided with LaTeX.

```
22.12 \@namedef{captions\CurrentOption}{%
22.13   \def\prefacename{Vorwort}%
22.14   \def\refname{Literatur}%
22.15   \def\abstractname{Zusammenfassung}%
22.16   \def\bibname{Literaturverzeichnis}%
22.17   \def\chaptername{Kapitel}%
22.18   \def\appendixname{Anhang}%
22.19   \def\contentsname{Inhaltsverzeichnis}%    % oder nur: Inhalt
22.20   \def\listfigurename{Abbildungsverzeichnis}%
22.21   \def\listtablename{Tabellenverzeichnis}%
```

[17]The file described in this section has version number v2.6m and was last revised on 2004/02/20.

```
  \def\indexname{Index}%
  \def\figurename{Abbildung}%
  \def\tablename{Tabelle}%                  % oder: Tafel
  \def\partname{Teil}%
  \def\enclname{Anlage(n)}%                 % oder: Beilage(n)
  \def\ccname{Verteiler}%                   % oder: Kopien an
  \def\headtoname{An}%
  \def\pagename{Seite}%
  \def\seename{siehe}%
  \def\alsoname{siehe auch}%
  \def\proofname{Beweis}%
  \def\glossaryname{Glossar}%
  }
```

`\datengerman` The macro `\datengerman` redefines the command `\today` to produce German dates.

```
\def\month@ngerman{\ifcase\month\or
  Januar\or Februar\or M\"arz\or April\or Mai\or Juni\or
  Juli\or August\or September\or Oktober\or November\or Dezember\fi}
\def\datengerman{\def\today{\number\day.~\month@ngerman
    \space\number\year}}
```

`\dateanustrian` The macro `\datenaustrian` redefines the command `\today` to produce Austrian version of the German dates.

```
\def\datenaustrian{\def\today{\number\day.~\ifnum1=\month
  J\"anner\else \month@ngerman\fi \space\number\year}}
```

`\extrasngerman` `\extrasnaustrian` `\noextrasngerman` `\noextrasnaustrian` Either the macro `\extrasngerman` or the macros `\extrasnaustrian` will perform all the extra definitions needed for the German language. The macro `\noextrasngerman` is used to cancel the actions of `\extrasngerman`.

For German (as well as for Dutch) the `"` character is made active. This is done once, later on its definition may vary.

```
\initiate@active@char{"}
\@namedef{extras\CurrentOption}{%
  \languageshorthands{ngerman}}
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \bbl@activate{"}}
```

Don't forget to turn the shorthands off again.

```
\addto\noextrasngerman{\bbl@deactivate{"}}
```

In order for TEX to be able to hyphenate German words which contain 'ß' (in the `OT1` position `^^Y`) we have to give the character a nonzero `\lccode` (see Appendix H, the TEXbook).

```
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \babel@savevariable{\lccode25}%
  \lccode25=25}
```

The umlaut accent macro `\"` is changed to lower the umlaut dots. The redefinition is done with the help of `\umlautlow`.

```
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \babel@save\"\umlautlow}
\@namedef{noextras\CurrentOption}{\umlauthigh}
```

The current version of the 'new' German hyphenation patterns (`dehyphn.tex` is to be used with `\lefthyphenmin` and `\righthyphenmin` set to 2.

```
\providehyphenmins{\CurrentOption}{\tw@\tw@}
```

For German texts we need to make sure that `\frenchspacing` is turned on.

```
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \bbl@frenchspacing}
\expandafter\addto\csname noextras\CurrentOption\endcsname{%
  \bbl@nonfrenchspacing}
```

The code above is necessary because we need an extra active character. This character is then used as indicated in table 4.

To be able to define the function of `"`, we first define a couple of 'support' macros.

`\dq` We save the original double quote character in `\dq` to keep it available, the math accent `\"` can now be typed as `"`.

```
\begingroup \catcode`\"12
\def\x{\endgroup
  \def\@SS{\mathchar"7019 }
  \def\dq{"}}
\x
```

Now we can define the doublequote macros: the umlauts,

```
\declare@shorthand{ngerman}{"a}{\textormath{\"{a}\allowhyphens}{\ddot a}}
\declare@shorthand{ngerman}{"o}{\textormath{\"{o}\allowhyphens}{\ddot o}}
\declare@shorthand{ngerman}{"u}{\textormath{\"{u}\allowhyphens}{\ddot u}}
\declare@shorthand{ngerman}{"A}{\textormath{\"{A}\allowhyphens}{\ddot A}}
\declare@shorthand{ngerman}{"O}{\textormath{\"{O}\allowhyphens}{\ddot O}}
\declare@shorthand{ngerman}{"U}{\textormath{\"{U}\allowhyphens}{\ddot U}}
```

tremas,

```
\declare@shorthand{ngerman}{"e}{\textormath{\"{e}}{\ddot e}}
\declare@shorthand{ngerman}{"E}{\textormath{\"{E}}{\ddot E}}
\declare@shorthand{ngerman}{"i}{\textormath{\"{\i}}%
                              {\ddot\imath}}
\declare@shorthand{ngerman}{"I}{\textormath{\"{I}}{\ddot I}}
```

german es-zet (sharp s),

```
\declare@shorthand{ngerman}{"s}{\textormath{\ss}{\@SS{}}}
\declare@shorthand{ngerman}{"S}{\SS}
\declare@shorthand{ngerman}{"z}{\textormath{\ss}{\@SS{}}}
\declare@shorthand{ngerman}{"Z}{SZ}
```

german and french quotes,

```
\declare@shorthand{ngerman}{"`}{\glqq}
\declare@shorthand{ngerman}{"'}{\grqq}
```

```
\declare@shorthand{ngerman}{"<}{\flqq}
\declare@shorthand{ngerman}{">}{\frqq}
```

and some additional commands:

```
\declare@shorthand{ngerman}{"-}{\nobreak\-\bbl@allowhyphens}
\declare@shorthand{ngerman}{"|}{%
  \textormath{\penalty\@M\discretionary{-}{}{\kern.03em}%
              \allowhyphens}{}}
\declare@shorthand{ngerman}{""}{\hskip\z@skip}
\declare@shorthand{ngerman}{"~}{\textormath{\leavevmode\hbox{-}}{-}}
\declare@shorthand{ngerman}{"=}{\penalty\@M-\hskip\z@skip}
```

`\mdqon` `\mdqoff` All that's left to do now is to define a couple of commands for reasons of compatibility with `german.sty`.

```
\def\mdqon{\shorthandon{"}}
\def\mdqoff{\shorthandoff{"}}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish\CurrentOption
⟨/code⟩
```

## 23 The Breton language

The file `breton.dtx`[18] defines all the language-specific macros for the Breton language.

There are not really typographic rules for the Breton language. It is a local language (it's one of the celtic languages) which is spoken in Brittany (West of France). So we have a synthesis between french typographic rules and english typographic rules. The characters `:`, `;`, `!` and `?` are made active in order to get a whitespace automatically before these characters.

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
⟨*code⟩
\LdfInit{breton}\captionsbreton
```

When this file is read as an option, i.e. by the `\usepackage` command, `breton` will be an 'unknown' language in which case we have to make it known. So we check for the existence of `\l@breton` to see whether we have to do something here.

```
\ifx\l@breton\@undefined
    \@nopatterns{Breton}
    \adddialect\l@breton0\fi
```

The next step consists of defining commands to switch to the English language. The reason for this is that a user might want to switch back and forth between languages.

`\captionsbreton` The macro `\captionsbreton` defines all strings used in the four standard document classes provided with LaTeX.

```
\addto\captionsbreton{%
  \def\prefacename{Rakskrid}%
  \def\refname{Daveenno\`u}%
  \def\abstractname{Dvierra\~n}%
  \def\bibname{Lennadurezh}%
  \def\chaptername{Pennad}%
  \def\appendixname{Stagadenn}%
  \def\contentsname{Taolenn}%
  \def\listfigurename{Listenn ar Figurenno\`u}%
  \def\listtablename{Listenn an taolenno\`u}%
  \def\indexname{Meneger}%
  \def\figurename{Figurenn}%
  \def\tablename{Taolenn}%
  \def\partname{Lodenn}%
  \def\enclname{Diello\`u kevret}%
  \def\ccname{Eilskrid da}%
  \def\headtoname{evit}
  \def\pagename{Pajenn}%
  \def\seename{Gwelout}%
```

[18]The file described in this section has version number v1.0h and was last revised on 2005/03/29.

```
  \def\alsoname{Gwelout ivez}%
  \def\proofname{Proof}%  <-- needs translation
  \def\glossaryname{Glossary}% <-- Needs translation
}
```

\datebreton The macro `\datebreton` redefines the command `\today` to produce Breton dates.

```
\def\datebreton{%
  \def\today{\ifnum\day=1\relax 1\/$^{\rm a\tilde{n}}$\else
    \number\day\fi \space a\space viz\space\ifcase\month\or
    Genver\or C'hwevrer\or Meurzh\or Ebrel\or Mae\or Mezheven\or
    Gouere\or Eost\or Gwengolo\or Here\or Du\or Kerzu\fi
    \space\number\year}}
```

\extrasbreton \noextrasbreton The macro `\extrasbreton` will perform all the extra definitions needed for the Breton language. The macro `\noextrasbreton` is used to cancel the actions of `\extrasbreton`.

The category code of the characters `:`, `;`, `!` and `?` is made `\active` to insert a little white space.

```
\initiate@active@char{:}
\initiate@active@char{;}
\initiate@active@char{!}
\initiate@active@char{?}
```

We specify that the breton group of shorthands should be used.

```
\addto\extrasbreton{\languageshorthands{breton}}
```

These characters are 'turned on' once, later their definition may vary.

```
\addto\extrasbreton{%
  \bbl@activate{:}\bbl@activate{;}%
  \bbl@activate{!}\bbl@activate{?}}
```

Don't forget to turn the shorthands off again.

```
\addto\noextrasbreton{%
 \bbl@deactivate{:}\bbl@deactivate{;}%
 \bbl@deactivate{!}\bbl@deactivate{?}}
```

The last thing `\extrasbreton` needs to do is to make sure that `\frenchspacing` is in effect. If this is not the case the execution of `\noextrasbreton` will switch it of again.

```
\addto\extrasbreton{\bbl@frenchspacing}
\addto\noextrasbreton{\bbl@nonfrenchspacing}
```

\breton@sh@;@ We have to reduce the amount of white space before `;`, `:` and `!` when the user types a space in front of these characters. This should only happen outside mathmode, hence the test with `\ifmmode`.

```
\declare@shorthand{breton}{;}{%
    \ifmmode
      \string;\space
    \else\relax
```

In horizontal mode we check for the presence of a 'space' and replace it by a `\thinspace`.

```
    \ifhmode
      \ifdim\lastskip>\z@
        \unskip\penalty\@M\thinspace
      \fi
    \fi
    \string;\space
  \fi}%
```

`\breton@sh@:@` `\breton@sh@!@` Because these definitions are very similar only one is displayed in a way that the definition can be easily checked.

```
\declare@shorthand{breton}{:}{%
  \ifmmode\string:\space
  \else\relax
    \ifhmode
      \ifdim\lastskip>\z@\unskip\penalty\@M\thinspace\fi
    \fi
    \string:\space
  \fi}
\declare@shorthand{breton}{!}{%
  \ifmmode\string!\space
  \else\relax
    \ifhmode
      \ifdim\lastskip>\z@\unskip\penalty\@M\thinspace\fi
    \fi
    \string!\space
  \fi}
```

`\breton@sh@?@` For the question mark something different has to be done. In this case the amount of white space that replaces the space character depends on the dimensions of the font.

```
\declare@shorthand{breton}{?}{%
  \ifmmode
    \string?\space
  \else\relax
    \ifhmode
      \ifdim\lastskip>\z@
        \unskip
        \kern\fontdimen2\font
        \kern-1.4\fontdimen3\font
      \fi
    \fi
    \string?\space
  \fi}
```

All that is left to do now is provide the breton user with some extra utilities. Some definitions for special characters.

```
\DeclareTextSymbol{\at}{OT1}{64}
```

```
\DeclareTextSymbol{\at}{T1}{64}
\DeclareTextSymbolDefault{\at}{OT1}
\DeclareTextSymbol{\boi}{OT1}{92}
\DeclareTextSymbol{\boi}{T1}{16}
\DeclareTextSymbolDefault{\boi}{OT1}
\DeclareTextSymbol{\circonflexe}{OT1}{94}
\DeclareTextSymbol{\circonflexe}{T1}{2}
\DeclareTextSymbolDefault{\circonflexe}{OT1}
\DeclareTextSymbol{\tild}{OT1}{126}
\DeclareTextSymbol{\tild}{T1}{3}
\DeclareTextSymbolDefault{\tild}{OT1}
\DeclareTextSymbol{\degre}{OT1}{23}
\DeclareTextSymbol{\degre}{T1}{6}
\DeclareTextSymbolDefault{\degre}{OT1}
```

The following macros are used in the redefinition of `\^` and `\"` to handle the letter i.

```
\AtBeginDocument{%
  \DeclareTextCompositeCommand{\^}{OT1}{i}{\^\i}
  \DeclareTextCompositeCommand{\"}{OT1}{i}{\"\i}}
```

And some more macros for numbering.

```
\def\kentan{1\/${}^{\rm a\tilde{n}}$}
\def\eil{2\/${}^{\rm l}$}
\def\re{\/${}^{\rm re}$}
\def\trede{3\re}
\def\pevare{4\re}
\def\vet{\/${}^{\rm vet}$}
\def\pempvet{5\vet}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{breton}
⟨/code⟩
```

## 24 The Welsh language

The file `welsh.dtx`[19] defines all the language definition macros for the Welsh language.

For this language currently no special definitions are needed or available.

The macro `\ldf@init` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
⟨*code⟩
\LdfInit{welsh}{captionswelsh}
```

When this file is read as an option, i.e. by the `\usepackage` command, `welsh` could be an 'unknown' language in which case we have to make it known. So we check for the existence of `\l@welsh` to see whether we have to do something here.

```
\ifx\undefined\l@welsh
  \@nopatterns{welsh}
  \adddialect\l@welsh0\fi
```

The next step consists of defining commands to switch to (and from) the Welsh language.

`\welshhyphenmins` This macro is used to store the correct values of the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`.

```
\providehyphenmins{\CurrentOption}{\tw@\thr@@}
```

`\captionswelsh` The macro `\captionswelsh` defines all strings used in the four standard documentclasses provided with LaTeX.

```
\def\captionswelsh{%
  \def\prefacename{Rhagair}%
  \def\refname{Cyfeiriadau}%
  \def\abstractname{Crynodeb}%
  \def\bibname{Llyfryddiaeth}%
  \def\chaptername{Pennod}%
  \def\appendixname{Atodiad}%
  \def\contentsname{Cynnwys}%
  \def\listfigurename{Rhestr Ddarluniau}%
  \def\listtablename{Rhestr Dablau}%
  \def\indexname{Mynegai}%
  \def\figurename{Darlun}%
  \def\tablename{Taflen}%
  \def\partname{Rhan}%
  \def\enclname{amgae\"edig}%
  \def\ccname{cop\"\i au}%
  \def\headtoname{At}%  % 'at' on letters meaning 'to ( a person)'
                        % 'to (a place)' is 'i' in Welsh
  \def\pagename{tudalen}%
  \def\seename{gweler}%
  \def\alsoname{gweler hefyd}%
```

[19]The file described in this section has version number v1.0d and was last revised on 2005/03/31.

```
  \def\proofname{Prawf}%
  \def\glossaryname{Rhestr termau}%
  }
```

\datewelsh The macro `\datewelsh` redefines the command `\today` to produce welsh dates.

```
\def\datewelsh{%
  \def\today{\ifnum\day=1\relax 1\/$^{\mathrm{a\tilde{n}}}$\else
    \number\day\fi\space\ifcase\month\or
    Ionawr\or Chwefror\or Mawrth\or Ebrill\or
    Mai\or Mehefin\or Gorffennaf\or Awst\or
    Medi\or Hydref\or Tachwedd\or Rhagfyr\fi
  \space\number\year}}
```

\extraswelsh \noextraswelsh The macro `\extraswelsh` will perform all the extra definitions needed for the welsh language. The macro `\noextraswelsh` is used to cancel the actions of `\extraswelsh`. For the moment these macros are empty but they are defined for compatibility with the other language definition files.

```
\addto\extraswelsh{}
\addto\noextraswelsh{}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{welsh}
⟨/code⟩
```

## 25 The Irish language

The file `irish.dtx`[20] defines all the language definition macros for the Irish language.

For this language currently no special definitions are needed or available.

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
⟨*code⟩
\LdfInit{irish}\captionsirish
```

When this file is read as an option, i.e. by the `\usepackage` command, `irish` could be an 'unknown' language in which case we have to make it known. So we check for the existence of `\l@irish` to see whether we have to do something here.

```
\ifx\l@irish\@undefined
  \@nopatterns{irish}
  \adddialect\l@irish0\fi
```

The next step consists of defining commands to switch to (and from) the Irish language.

`\irishhyphenmins` This macro is used to store the correct values of the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`.

```
\providehyphenmins{\CurrentOption}{\tw@\thr@@}
```

`\captionsirish` The macro `\captionsirish` defines all strings used in the four standard documentclasses provided with LaTeX.

```
\addto\captionsirish{%
  \def\prefacename{R\'eamhr\'a}%    <-- also "Brollach"
  \def\refname{Tagairt\'{\i}}%
  \def\abstractname{Achoimre}%
  \def\bibname{Leabharliosta}%
  \def\chaptername{Caibidil}%
  \def\appendixname{Aguis\'{\i}n}%
  \def\contentsname{Cl\'ar \'Abhair}%
  \def\listfigurename{L\'ear\'aid\'{\i}}%
  \def\listtablename{T\'abla\'{\i}}%
  \def\indexname{Inn\'eacs}%
  \def\figurename{L\'ear\'aid}%
  \def\tablename{T\'abla}%
  \def\partname{Cuid}%
  \def\enclname{faoi iamh}%
  \def\ccname{cc}%                 abrv. 'c\'oip chuig'
  \def\headtoname{Go}%
  \def\pagename{Leathanach}%
  \def\seename{f\'each}%
  \def\alsoname{f\'each freisin}%
  \def\proofname{Cruth\'unas}%
```

[20] The file described in this section has version number v1.0h and was last revised on 2005/03/30. A contribution was made by Marion Gunn.

```
  \def\glossaryname{Glossary}% <-- Needs translation
  }
```

`\dateirish` The macro `\dateirish` redefines the command `\today` to produce Irish dates.

```
\def\dateirish{%
  \def\today{%
    \number\day\space \ifcase\month\or
    Ean\'air\or Feabhra\or M\'arta\or Aibre\'an\or
    Bealtaine\or Meitheamh\or I\'uil\or L\'unasa\or
    Me\'an F\'omhair\or Deireadh F\'omhair\or
    M\'{\i} na Samhna\or M\'{\i} na Nollag\fi
    \space \number\year}}
```

`\extrasirish` `\noextrasirish` The macro `\extrasirish` will perform all the extra definitions needed for the Irish language. The macro `\noextrasirish` is used to cancel the actions of `\extrasirish`. For the moment these macros are empty but they are defined for compatibility with the other language definition files.

```
\addto\extrasirish{}
\addto\noextrasirish{}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{irish}
⟨/code⟩
```

## 26 The Scottish language

The file `scottish.dtx`[21] defines all the language definition macros for the Scottish language.

For this language currently no special definitions are needed or available.

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
⟨*code⟩
\LdfInit{scottish}\captionsscottish
```

When this file is read as an option, i.e. by the `\usepackage` command, `scottish` could be an 'unknown' language in which case we have to make it known. So we check for the existence of `\l@scottish` to see whether we have to do something here.

```
\ifx\l@scottish\@undefined
  \@nopatterns{scottish}
  \adddialect\l@scottish0\fi
```

The next step consists of defining commands to switch to (and from) the Scottish language.

`\captionsscottish` The macro `\captionsscottish` defines all strings used in the four standard documentclasses provided with LaTeX.

```
\addto\captionsscottish{%
  \def\prefacename{Preface}%    <-- needs translation
  \def\refname{Iomraidh}%
  \def\abstractname{Br\`{\i}gh}%
  \def\bibname{Leabhraichean}%
  \def\chaptername{Caibideil}%
  \def\appendixname{Ath-sgr`{\i}obhadh}%
  \def\contentsname{Cl\`ar-obrach}%
  \def\listfigurename{Liosta Dhealbh }%
  \def\listtablename{Liosta Chl\`ar}%
  \def\indexname{Cl\`ar-innse}%
  \def\figurename{Dealbh}%
  \def\tablename{Cl\`ar}%
  \def\partname{Cuid}%
  \def\enclname{a-staigh}%
  \def\ccname{lethbhreac gu}%
  \def\headtoname{gu}%
  \def\pagename{t.d.}%             abrv. `taobh duilleag'
  \def\seename{see}%    <-- needs translation
  \def\alsoname{see also}%    <-- needs translation
  \def\proofname{Proof}%    <-- needs translation
  \def\glossaryname{Glossary}% <-- Needs translation
}
```

[21] The file described in this section has version number v1.0g and was last revised on 2005/03/31. A contribution was made by Fraser Grant (FRASER@CERNVM).

\datescottish The macro `\datescottish` redefines the command `\today` to produce Scottish dates.

```
\def\datescottish{%
  \def\today{%
    \number\day\space \ifcase\month\or
    am Faoilteach\or an Gearran\or am M\`art\or an Giblean\or
    an C\`eitean\or an t-\`Og mhios\or an t-Iuchar\or
    L\`unasdal\or an Sultuine\or an D\`amhar\or
    an t-Samhainn\or an Dubhlachd\fi
    \space \number\year}}
```

\extrasscottish
\noextrasscottish The macro `\extrasscottish` will perform all the extra definitions needed for the Scottish language. The macro `\noextrasscottish` is used to cancel the actions of `\extrasscottish`. For the moment these macros are empty but they are defined for compatibility with the other language definition files.

```
\addto\extrasscottish{}
\addto\noextrasscottish{}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{scottish}
⟨/code⟩
```

## 27 The Greek language

The file `greek.dtx`[22] defines all the language definition macros for the Greek language, i.e., as it used today with only one accent, and the attribute *πολυτονικό* ("Polutoniko") for typesetting greek text with all accents. This separation arose out of the need to simplify things, for only very few people will be really interested to typeset polytonic Greek text.

`\greektext` `\latintext` The commands `\greektext` and `\latintext` can be used to switch to greek or latin fonts. These are declarations.

`\textgreek` `\textlatin` `\textol` The commands `\textgreek` and `\textlatin` both take one argument which is then typeset using the requested font encoding. The command `\greekol` switches to the greek outline font family, while the command `\textol` typests a short text in outline font. A number of extra greek characters are made available through the added text commands `\stigma`, `\qoppa`, `\sampi`, `\ddigamma`, `\Digamma`, `\euro`, `\permill`, and `\vardigamma`.

### 27.1 Typing conventions

Entering greek text can be quite difficult because of the many diacritical signs that need to be added for various purposes. The fonts that are used to typeset Greek make this a lot easier by offering a lot of ligatures. But in order for this to work, some characters need to be considered as letters. These characters are `<`, `>`, `~`, `` ` ``, `'`, `"` and `|`. Therefore their `\lccode` is changed when Greek is in effect. In order to let `\uppercase` give correct results, the `\uccode` of these characters is set to a non-existing character to make them disappear. Of course not all characters are needed when typesetting "modern" *μονοτονικό*. In that case we only need the `'` and `"` symbols which are treated in the proper way.

### 27.2 Greek numbering

The Greek alphabetical numbering system, like the Roman one, is still used in everyday life for short enumerations. Unfortunately most Greeks don't know how to write Greek numbers bigger than 20 or 30. Nevertheless, in official editions of the last century and beginning of this century this numbering system was also used for dates and numbers in the range of several thousands. Nowadays this numbering system is primary used by the Eastern Orthodox Church and by certain scholars. It is hence necessary to be able to typeset any Greek numeral up to 999 999. Here are the conventions:

- There is no Greek numeral for any number less than or equal to 0.
- Numbers from 1 to 9 are denoted by letters alpha, beta, gamma, delta, epsilon, stigma, zeta, eta, theta, followed by a mark similar to the math-

[22]The file described in this section has version number v1.3l and was last revised on 2005/03/30. The original author is Apostolos Syropoulos (`apostolo@platon.ee.duth.gr`), code from `kdgreek.sty` by David Kastrup `dak@neuroinformatik.ruhr-uni-bochum.de` was used to enhance the support for typesetting greek texts.

ematical symbol “prime”. (Nowadays instead of letter stigma the digraph sigma tau is used for number 6. Mainly because the letter stigma is not always available, so people opt to write down the first two letters of its name as an alternative. In our implementation we produce the letter stigma, not the digraph sigma tau.)

- Decades from 10 to 90 are denoted by letters iota, kappa, lambda, mu, nu, xi, omikron, pi, qoppa, again followed by the numeric mark. The qoppa used for this purpose has a special zig-zag form, which doesn’t resemble at all the original ‘q’-like qoppa.
- Hundreds from 100 to 900 are denoted by letters rho, sigma, tau, upsilon, phi, chi, psi, omega, sampi, followed by the numeric mark.
- Any number between 1 and 999 is obtained by a group of letters denoting the hundreds decades and units, followed by a numeric mark.
- To denote thousands one uses the same method, but this time the mark is placed in front of the letter, and under the baseline (it is inverted by 180 degrees). When a group of letters denoting thousands is followed by a group of letters denoting a number under 1000, then both marks are used.

`\greeknumeral` `\Greeknumeral`

Using these conventions one obtains numbers up to 999 999. The command `\greeknumeral` makes it possible to typeset Greek numerals. There is also an “uppercase” version of this macro: `\Greeknumeral`.

Another system which was in wide use only in Athens, could express any positive number. This system is implemented in package `athnum`.

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
27.1 ⟨*code⟩
27.2 \LdfInit\CurrentOption{captions\CurrentOption}
```

When the option polutonikogreek was used, redefine `\CurrentOption` to prevent problems later on.

```
27.3 \gdef\CurrentOption{greek}%
```

When this file is read as an option, i.e. by the `\usepackage` command, `greek` could be an ‘unknown’ language in which case we have to make it known. So we check for the existence of `\l@greek` to see whether we have to do something here.

```
27.4 \ifx\l@greek\@undefined
27.5   \@nopatterns{greek}
27.6   \adddialect\l@greek0\fi
```

Now we declare the `polutoniko` language attribute.

```
27.7 \bbl@declare@ttribute{greek}{polutoniko}{%
```

This code adds the expansion of `\extraspolutonikogreek` to `\extrasgreek` and changes the definition of `\today` for Greek to produce polytonic month names.

```
27.8   \expandafter\addto\expandafter\extrasgreek
27.9   \expandafter{\extraspolutonikogreek}%
```

```
  \let\captionsgreek\captionspolutonikogreek
  \let\gr@month\gr@c@month
```

We need to take some extra precautions in order not to break older documents which still use the old polutonikogreek option.

```
  \let\l@polutonikogreek\l@greek
  \let\datepolutonikogreek\dategreek
  \let\extraspolutonikogreek\extrasgreek
  \let\noextraspolutonikogreek\noextrasgreek
  }
```

Typesetting Greek texts implies that a special set of fonts needs to be used. The current support for greek uses the cb fonts created by Claudio Beccari[23]. The cb fonts provide all sorts of *font combinations*. In order to use these fonts we define the Local GReek encoding (LGR, see the file greek.fdd). We make sure that this encoding is known to LaTeX, and if it isn't we abort.

```
\InputIfFileExists{lgrenc.def}{%
  \message{Loading the definitions for the Greek font encoding}}{%
  \errhelp{I can't find the lgrenc.def file for the Greek fonts}%
  \errmessage{Since I do not know what the LGR encoding means^^J
    I can't typeset Greek.^^J
    I stop here, while you get a suitable lgrenc.def file}\@@end
}
```

Now we define two commands that offer the possibility to switch between Greek and Roman encodings.

\greektext The command \greektext will switch from Latin font encoding to the Greek font encoding. This assumes that the 'normal' font encoding is a Latin one. This command is a *declaration*, for shorter pieces of text the command \textgreek should be used.

```
\DeclareRobustCommand{\greektext}{%
  \fontencoding{LGR}\selectfont
  \def\encodingdefault{LGR}}
```

\textgreek This command takes an argument which is then typeset using the requested font encoding. In order to avoid many encoding switches it operates in a local scope.

```
\DeclareRobustCommand{\textgreek}[1]{\leavevmode{\greektext #1}}
```

\textol A last aspect of the set of fonts provided with this version of support for typesetting Greek texts is that it contains an outline family. In order to make it available we define the command \textol.

```
\def\outlfamily{\usefont{LGR}{cmro}{m}{n}}
\DeclareTextFontCommand{\textol}{\outlfamily}
```

The next step consists in defining commands to switch to (and from) the Greek language.

[23]Apostolos Syropoulos wishes to thank him for his patience, collaboration, cooments and suggestions.

\greekhyphenmins This macro is used to store the correct values of the hyphenation parameters \lefthyphenmin and \righthyphenmin.

```
% Yannis Haralambous has suggested this value
\providehyphenmins{\CurrentOption}{\@ne\@ne}
```

\captionsgreek The macro \captionsgreek defines all strings used in the four standard document classes provided with LaTeX.

```
\addto\captionsgreek{%
  \def\prefacename{Pr'ologos}%
  \def\refname{Anafor'es}%
  \def\abstractname{Per'ilhyh}%
  \def\bibname{Bibliograf'ia}%
  \def\chaptername{Kef'alaio}%
  \def\appendixname{Par'arthma}%
  \def\contentsname{Perieq'omena}%
  \def\listfigurename{Kat'alogos Sqhm'atwn}%
  \def\listtablename{Kat'alogos Pin'akwn}%
  \def\indexname{Euret'hrio}%
  \def\figurename{Sq'hma}%
  \def\tablename{P'inakas}%
  \def\partname{M'eros}%
  \def\enclname{Sunhmm'ena}%
  \def\ccname{Koinopo'ihsh}%
  \def\headtoname{Pros}%
  \def\pagename{Sel'ida}%
  \def\seename{bl'epe}%
  \def\alsoname{bl'epe ep'ishs}%
  \def\proofname{Ap'odeixh}%
  \def\glossaryname{Glwss'ari}%
  }
```

\captionspolutonikogreek For texts written in the *πολυτονικό* (polytonic greek) the translations are the same as above, but some words are spelled differently. For now we just add extra definitions to \captionsgreek in order to override the earlier definitions.

```
\let\captionspolutonikogreek\captionsgreek
\addto\captionspolutonikogreek{%
  \def\refname{>Anafor`es}%
  \def\indexname{E<uret'hrio}%
  \def\figurename{Sq~hma}%
  \def\headtoname{Pr`os}%
  \def\alsoname{bl'epe >ep'ishs}%
  \def\proofname{>Ap'odeixh}%
}
```

\gr@month \dategreek The macro \dategreek redefines the command \today to produce greek dates. The name of the month is now produced by the macro \gr@month since it is needed in the definition of the macro \Grtoday.

```
\def\gr@month{%
  \ifcase\month\or
    Ianouar'iou\or Febrouar'iou\or Mart'iou\or April'iou\or
    Ma'"iou\or Ioun'iou\or Ioul'iou\or Augo'ustou\or
    Septembr'iou\or Oktwbr'iou\or Noembr'iou\or Dekembr'iou\fi}
\def\dategreek{%
  \def\today{\number\day \space \gr@month\space \number\year}}
```

\gr@c@greek

```
\def\gr@c@month{%
  \ifcase\month\or >Ianouar'iou\or
    Febrouar'iou\or Mart'iou\or >April'iou\or Ma"'iou\or
    >Ioun'iou\or  >Ioul'iou\or A>ugo'ustou\or Septembr'iou\or
    >Oktwbr'iou\or Noembr'iou\or Dekembr'iou\fi}
```

\Grtoday The macro \Grtoday produces the current date, only that the month and the day are shown as greek numerals instead of arabic as it is usually the case.

```
\def\Grtoday{%
  \expandafter\Greeknumeral\expandafter{\the\day}\space
  \gr@c@month \space
  \expandafter\Greeknumeral\expandafter{\the\year}}
```

\extrasgreek \noextrasgreek The macro \extrasgreek will perform all the extra definitions needed for the Greek language. The macro \noextrasgreek is used to cancel the actions of \extrasgreek. For the moment these macros switch the fontencoding used and the definition of the internal macros \@alph and \@Alph because in Greek we do use the Greek numerals.

```
\addto\extrasgreek{\greektext}
\addto\noextrasgreek{\latintext}
```

\gr@ill@value When the argument of \greeknumeral has a value outside of the acceptable bounds ($0 < x < 999999$) a warning will be issued (and nothing will be printed).

```
\def\gr@ill@value#1{%
  \PackageWarning{babel}{Illegal value (#1) for greeknumeral}}
```

\anw@true \anw@false \anw@print When a a large number with three *trailing* zero's is to be printed those zeros *and* the numeric mark need to be discarded. As each 'digit' is processed by a separate macro *and* because the processing needs to be expandable we need some helper macros that help remember to *not* print the numeric mark (\anwtonos).

The command \anw@false switches the printing of the numeric mark off by making \anw@print expand to nothing. The command \anw@true (re)enables the printing of the numeric marc. These macro's need to be robust in order to prevent improper expansion during writing to files or during \uppercase.

```
\DeclareRobustCommand\anw@false{%
  \DeclareRobustCommand\anw@print{}}
\DeclareRobustCommand\anw@true{%
  \DeclareRobustCommand\anw@print{\anwtonos}}
\anw@true
```

`\greeknumeral` The command `\greeknumeral` needs to be *fully* expandable in order to get the right information in auxiliary files. Therefore we use a big `\if`-construction to check the value of the argument and start the parsing at the right level.

```
\def\greeknumeral#1{%
```

If the value is negative or zero nothing is printed and a warning is issued.

```
  \ifnum#1<\@ne\space\gr@ill@value{#1}%
  \else
    \ifnum#1<10\expandafter\gr@num@i\number#1%
    \else
      \ifnum#1<100\expandafter\gr@num@ii\number#1%
      \else
```

We use the available shorthands for 1.000 (`\@m`) and 10.000 (`\@M`) to save a few tokens.

```
        \ifnum#1<\@m\expandafter\gr@num@iii\number#1%
        \else
          \ifnum#1<\@M\expandafter\gr@num@iv\number#1%
          \else
            \ifnum#1<100000\expandafter\gr@num@v\number#1%
            \else
              \ifnum#1<1000000\expandafter\gr@num@vi\number#1%
              \else
```

If the value is too large, nothing is printed and a warning is issued.

```
                \space\gr@ill@value{#1}%
              \fi
            \fi
          \fi
        \fi
      \fi
    \fi
  \fi
}
```

`\Greeknumeral` The command `\Greeknumeral` prints uppercase greek numerals. The parsing is performed by the macro `\greeknumeral`.

```
\def\Greeknumeral#1{%
  \expandafter\MakeUppercase\expandafter{\greeknumeral{#1}}}
```

`\greek@alph` `\greek@Alph` In the previous release of this language definition the commands `\greek@aplh` and `\greek@Alph` were kept just for reasons of compatibility. Here again they become meaningful macros. They are definited in a way that even page numbering with greek numerals is possible. Since the macros `\@alph` and `\@Alph` will lose their original meaning while the Greek option is active, we must save their original value. macros `\@alph`

```
\let\latin@alph\@alph
\let\latin@Alph\@Alph
```

Then we define the Greek versions; the additional `\expandafter`s are needed in order to make sure the table of contents will be correct, e.g., when we have appendixes.

```
27.117 \def\greek@alph#1{\expandafter\greeknumeral\expandafter{\the#1}}
27.118 \def\greek@Alph#1{\expandafter\Greeknumeral\expandafter{\the#1}}
```

Now we can set up the switching.

```
27.119 \addto\extrasgreek{%
27.120   \let\@alph\greek@alph
27.121   \let\@Alph\greek@Alph}
27.122 \addto\noextrasgreek{%
27.123   \let\@alph\latin@alph
27.124   \let\@Alph\latin@Alph}
```

`\greek@roman` `\greek@Roman` To prevent roman numerals being typeset in greek letters we need to adopt the internal LaTeX commands `\@roman` and `\@Roman`. **Note that this may cause errors where roman ends up in a situation where it needs to be expanded; problems are known to exist with the AMS document classes.**

```
27.125 \let\latin@roman\@roman
27.126 \let\latin@Roman\@Roman
27.127 \def\greek@roman#1{\textlatin{\latin@roman{#1}}}
27.128 \def\greek@Roman#1{\textlatin{\latin@Roman{#1}}}
27.129 \addto\extrasgreek{%
27.130   \let\@roman\greek@roman
27.131   \let\@Roman\greek@Roman}
27.132 \addto\noextrasgreek{%
27.133   \let\@roman\latin@roman
27.134   \let\@Roman\latin@Roman}
```

`\greek@amp` `\ltx@amp` The greek fonts do not contain an ampersand, so the LaTeX command `\&` doesn't give the expected result if we do not do something about it.

```
27.135 \let\ltx@amp\&
27.136 \def\greek@amp{\textlatin{\ltx@amp}}
27.137 \addto\extrasgreek{\let\&\greek@amp}
27.138 \addto\noextrasgreek{\let\&\ltx@amp}
```

What is left now is the definition of a set of macros to produce the various digits.

`\gr@num@i` `\gr@num@ii` `\gr@num@iii` As there is no representation for 0 in this system the zeros are simply discarded. When we have a large number with three *trailing* zero's also the numeric mark is discarded. Therefore these macros need to pass the information to each other about the (non-)translation of a zero.

```
27.139 \def\gr@num@i#1{%
27.140   \ifcase#1\or a\or b\or g\or d\or e\or \stigma\or z\or h\or j\fi
27.141   \ifnum#1=\z@\else\anw@true\fi\anw@print}
27.142 \def\gr@num@ii#1{%
27.143   \ifcase#1\or i\or k\or l\or m\or n\or x\or o\or p\or \qoppa\fi
27.144   \ifnum#1=\z@\else\anw@true\fi\gr@num@i}
```

```
\def\gr@num@iii#1{%
  \ifcase#1\or r\or sv\or t\or u\or f\or q\or y\or w\or \sampi\fi
  \ifnum#1=\z@\anw@false\else\anw@true\fi\gr@num@ii}
```

`\gr@num@iv` `\gr@num@v` `\gr@num@vi` The first three 'digits' always have the numeric mark, except when one is discarded because it's value is zero.

```
\def\gr@num@iv#1{%
  \ifnum#1=\z@\else\katwtonos\fi
  \ifcase#1\or a\or b\or g\or d\or e\or \stigma\or z\or h\or j\fi
  \gr@num@iii}
\def\gr@num@v#1{%
  \ifnum#1=\z@\else\katwtonos\fi
  \ifcase#1\or i\or k\or l\or m\or n\or x\or o\or p\or \qoppa\fi
  \gr@num@iv}
\def\gr@num@vi#1{%
  \katwtonos
  \ifcase#1\or r\or sv\or t\or u\or f\or q\or y\or w\or \sampi\fi
  \gr@num@v}
```

`\greek@tilde` In greek typesetting we need a number of characters with more than one accent. In the underlying family of fonts (the `cb` fonts) this is solved using Knuth's ligature mechanism. Characters we need to have ligatures with are the tilde, the acute and grave accent characters, the rough and smooth breathings, the subscript, and the double quote character. In text input the `~` is normaly used to produce an unbreakable space. The command `\~` normally produces a tilde accent. For polytonic Greek we change the definition of `\~` to produce the tilde character itself, making sure it has category code 12.

```
\begingroup
  \@ifundefined{active@char\string!}{}{\catcode`!=12\relax}
  \catcode`\~=12
  \lccode`\!=`\~
  \lowercase{\def\x{\endgroup
      \def\greek@tilde{!}}\x}
\addto\extraspolutonikogreek{%
  \babel@save\~\let\~\greek@tilde}
```

In order to get correct hyphenation we need to set the lower case code of a number of characters. The 'v' character has a special usage for the `cb` fonts: in fact this ligature mechanism detects the end of a word and assures that a final sigma is typeset with the proper sign wich is different from that of an initial or medial sigma; the 'v 'after an *isolated* sigma fools the ligature mechanism in order to typeset $\sigma$ in place of $\varsigma$. Because of this we make sure its lowercase code is not changed. For "modern" greek we have to deal only with `'` and `"` and so things are easy.

```
\addto\extrasgreek{%
  \babel@savevariable{\lccode`v}\lccode`v=`v%
  \babel@savevariable{\lccode`\'}\lccode`\'=`\'%
  \babel@savevariable{\lccode`\"}\lccode`\"=`\"}
```

```
\addto\extraspolutonikogreek{%
  \babel@savevariable{\lccode`\<}\lccode`\<=`\<%
  \babel@savevariable{\lccode`\>}\lccode`\>=`\>%
  \babel@savevariable{\lccode`\~}\lccode`\~=`\~%
  \babel@savevariable{\lccode`\|}\lccode`\|=`\|%
  \babel@savevariable{\lccode`\`}\lccode`\`=`\`}
```

And in order to get rid of all accents and breathings when a string is `\uppercase`d we also change a number of uppercase codes.

```
\addto\extrasgreek{%
  \babel@savevariable{\uccode`\"}\uccode`\"=`\"%
  \babel@savevariable{\uccode`\'}\uccode`\'=159} %% 159 == ^^9f
\addto\extraspolutonikogreek{%
  \babel@savevariable{\uccode`\~}\uccode`\~=159%
  \babel@savevariable{\uccode`\>}\uccode`\>=159%
  \babel@savevariable{\uccode`\<}\uccode`\<=159%
  \babel@savevariable{\uccode`\|}\uccode`\|=`\|%
  \babel@savevariable{\uccode`\`}\uccode`\`=159}
```

For this to work we make the character `^^9f` a shorthand that expands to nothing. In order for this to work we need to make a character look like `^^9f` in TEX's eyes. The trick is to have another character and assign it a different lowercase code. The execute the macros needed in a `\lowercase` environment. Usually the tile `~` character is used for such purposes. Before we do this we save it's original lowercase code to restore it once we're done.

```
\@tempcnta=\lccode`\~
\lccode`\~=159
\lowercase{%
  \initiate@active@char{~}%
  \declare@shorthand{greek}{~}{}}
\lccode`\~=\@tempcnta
```

We can also make the tilde character itself expand to a tilde with category code 12 to make the typing of texts easier.

```
\addto\extraspolutonikogreek{\languageshorthands{greek}}%
\declare@shorthand{greek}{~}{\greek@tilde}
```

We now define a few symbols which are used in the typesetting of greek numerals, as well as some other symbols which are usefull, such as the $\epsilon\upsilon\rho\omega$ symbol, etc.

```
\DeclareTextCommand{\anwtonos}{LGR}{\char"FE\relax}
\DeclareTextCommand{\katwtonos}{LGR}{\char"FF\relax}
\DeclareTextCommand{\qoppa}{LGR}{\char"12\relax}
\DeclareTextCommand{\stigma}{LGR}{\char"06\relax}
\DeclareTextCommand{\sampi}{LGR}{\char"1B\relax}
\DeclareTextCommand{\Digamma}{LGR}{\char"C3\relax}
\DeclareTextCommand{\ddigamma}{LGR}{\char"93\relax}
\DeclareTextCommand{\vardigamma}{LGR}{\char"07\relax}
\DeclareTextCommand{\euro}{LGR}{\char"18\relax}
\DeclareTextCommand{\permill}{LGR}{\char"19\relax}
```

Since the ~ cannot be used to produce an unbreakable white space we must redefine at least the commands `\fnum@figure` and `\fnum@table` so they do not produce a ~ instead of white space.

```
%\def\fnum@figure{\figurename\nobreakspace\thefigure}
%\def\fnum@table{\tablename\nobreakspace\thetable}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{\CurrentOption}
⟨/code⟩
```

## 28 The French language

The file `frenchb.dtx`[24], derived from `frenchy.sty`, defines all the language definition macros for the French language.

Customization for the French language is achieved following the book "Lexique des règles typographiques en usage à l'Imprimerie nationale" troisième édition (1994), ISBN-2-11-081075-0.

This file has been designed to be used with LaTeX $2_\varepsilon$, LaTeX-2.09 and PlainTeX formats. If you are still using LaTeX-2.09, you *should* consider switching to LaTeX $2_\varepsilon$!

The command `\selectlanguage{french}` switches to the French language [25], with the following effects:

1. French hyphenation patterns are made active;
2. 'double punctuation' is made active for correct spacing in French;
3. `\today` prints the date in French;
4. the caption names are translated into French (LaTeX only);
5. the default items in itemize environment are set to '–' instead of •, and all vertical spacing and glue is deleted, a hook to reset standard LaTeX spacing is provided (`\FrenchItemizeSpacingfalse`); it is possible to change '–' to something else ('—' for instance) by redefining `\FrenchLabelItem`; apart from the global hook `\FrenchLabelItem`, it is also possible to change the 'labelitems' at any level (1 to 4) in French, using the standard LaTeX syntax, for instance: `\renewcommand{\labelitemii}{\textbullet}`; in order to be effective in French, the redefinitions have to be made when French is the current language (i.e. *after* the `\begin{document}`), the changes are saved when switching to another language and will be remembered of, when switching back to French;
6. vertical spacing in general LaTeX lists is shortened, a hook to reset standard LaTeX settings is provided (`\FrenchListSpacingfalse`) ;
7. the first paragraph of each section is indented (LaTeX only);
8. the space after `\dots` is removed in French.

Some commands are provided in `frenchb` to make typesetting easier:

1. French quotation marks can be entered using the commands `\og` and `\fg` which work in LaTeX $2_\varepsilon$, LaTeX-2.09 and PlainTeX, their appearance depending on what is available to draw them; if you use LaTeX $2_\varepsilon$ *and* `T1`-encoding you can also enter them as `<<~French quotation marks~>>` but then *don't*

[24]The file described in this section has version number ? and was last revised on ?.

[25]`\selectlanguage{francais}` and `\selectlanguage{frenchb}` are kept for backward compatibility but should no longer be used.

*forget* the unbreakable spaces, (`\og` and `\fg` provide for correct line breaks and better horizontal spacing). `\og` and `\fg` can be used outside French, they provide then English quotes “ and ”.

2. A command `\up` is provided to typeset superscripts like `M\up{me}` (abbreviation for “Madame”), `1\up{er}` (for “premier”).
3. Family names should be typeset in small capitals and never be hyphenated, the macro `\bsc` (boxed small caps) does this, e.g., `Leslie~\bsc{Lamport}` will produce Leslie LAMPORT.
4. Commands `\primo`, `\secundo`, `\tertio` and `\quarto` may be used to enumerate in lists.
5. Abbreviations for “Numéro” and “numéro” are obtained via the commands `\No`, `\no`.
6. Two commands are provided to typeset the symbol for “degré”: `\degre` prints the raw character and `\degres` should be used to typeset temperatures (e.g., “`20~\degres C`” with an unbreakable space), or for alcohols’ strengths (e.g., “`45\degres`” with *no* space in French).
7. In math mode the comma has to be surrounded with braces to avoid a spurious space being inserted after it, in decimal numbers for instance (see the TeXbook p. 134). The command `\DecimalMathComma` makes the comma be an ordinary character *in French only* (no space added); as a counterpart, if `\DecimalMathComma` is active, an explicit space has to be added in lists and intervals: `$[0,\ 1]$`, `$(x,\ y)$`. `\StandardMathComma` switches back to the standard behaviour of the comma.
8. A command `\nombre` is provided to easily typeset numbers: it works both in text and in math mode: inputting `\nombre{3141,592653}` will format this number properly according to the current language (French or non-French): each slice of three digits will be separated either with a comma in English or with a space in French (if you prefer a thin space instead of a normal space, just add the command `\ThinSpaceInFrenchNumbers` to the preamble of your document, or to `frenchb.cfg`). The command `\nombre` is a contribution of Vincent Jalby using ideas of David Carlisle in comma.sty.
9. `frenchb` has been designed to take advantage of the `xspace` package if present: adding `\usepackage{xspace}` in the preamble will force macros like `\fg`, `\ier`, `\ieme`, `\dots`, . . . , to respect the spaces you type after them, for instance typing ‘`1\ier juin`’ will print ‘1$^{er}$ juin’ (no need for a forced space after `1\ier`).
10. Two commands, `\FrenchLayout` and `\StandardLayout` (to be used *only in the preamble*) are provided to unify the layout of multilingual documents (it mainly concerns lists) regardless the current language (see section 29.10 for details).

11. Two commands `\AddThinSpaceBeforeFootnotes` and `\FrenchFootnotes` enable to customize the layout of footnotes following the French IN's specifications; the first one, which can only be used *before* the `\begin{document}`, adds a thinspace in the running text before the number or symbol calling the footnote; the second one modifies the footnote's layout: it typesets leading numbers as '1. ' instead of '$^1$'. `\FrenchFootnotes` has no effect on footnotes numbered with symbols (as in the `\thanks` command).

    None of these commands are active by default, both can be best added to `frenchb.cfg` or to the preamble of the document. `\FrenchFootnotes` and `\StandardFootnotes`, which cancels the effect of `\FrenchFootnotes` for the text coming next, can be used anywhere in the document. The command `\StandardFootnotes` may be useful when some footnotes are numbered with letters (inside minipages for instance).

    `\AddThinSpaceBeforeFootnotes` and `\FrenchFootnotes` act on *all* footnotes whatever the current language is.

All commands previously available in `francais.ldf` have been included in `frenchb.ldf` for compatibility, sometimes with updated definitions.

The `french` package, by Bernard GAULLE, was not designed to run with babel (although the latest versions claim to be babel compatible), but rather as a standalone package for the French language. It provides many more functionalities (like `\lettrine`, `\sommaire`... ) not available in `frenchb`, at the cost of a much greater complexity and possible incompatibilities with other languages.

As `french` is known to produce the best layout available for French typography, I have borrowed many ideas from Bernard's file. I did my best to help users of both packages (`french` and `frenchb`) to exchange their sources files easily (see the example configuration file below in section 28), with one exception which affects the way French quotation marks are entered: `frenchb` uses *macros* (`\og` and `\fg`) while `french` uses active characters (`<<` and `>>`).

French typographic rules specify that some white space should be present before 'double punctuation' characters. These characters are `; ! ?` and `:`. In order to get this white space automatically, the category code of these characters is made `\active`. In French, the user *should* input these four characters preceded with a space, but as many people forget about it (even among native French writers!), the default behaviour of `frenchb` is to automatically add a `\thinspace` before '`;`' '`!`' '`?`' and a normal (unbreakable) space before '`:`' (this is the rule in French typography). It's up to the user to add or not a space *after* 'double punctuation' characters: usually a space is necessary, but not always (before a full point or a closing brace for instance), so this cannot done automatically.

In (rare) cases where no space should be added before a 'double punctuation', either use `\string; \string: \string! \string?` instead of `; : ! ?`, or switch locally to English. For instance you can type `C\string:TEX` or `\begin{otherlanguage}{english}{C:TEX}\end{otherlanguage}` to avoid the space before `:` in a MS-DOS path.

Some users dislike this automatic insertion of a space before 'double punctuation', and prefer to decide themselves whether a space should be added or not; so a hook `\NoAutoSpaceBeforeFDP` is provided: if this command is added (in file `frenchb.cfg`, or anywhere in a document) `frenchb` will respect your typing, and introduce a suitable space before 'double punctuation' *if and only if* a space is typed in the source file before those signs.

The command `\AutoSpaceBeforeFDP` switches back to the default behaviour of `frenchb`.

Once you have built your format, a good precaution would be to perform some basic tests about hyphenation in French. For LaTeX 2$_\varepsilon$ I suggest this:

- run the following file, with the encoding suitable for your machine (*my-encoding* will be `latin1` for UNIX machines and PCs running Windows, `applemac` for Macintoshs, or `cp850` for PCs running DOS).

  ```
  %%% Test file for French hyphenation.
  \documentclass{article}
  \usepackage[my-encoding]{inputenc}
  \usepackage[T1]{fontenc} % Use EC fonts for French
  %\usepackage{aeguill}    % Uncomment this line and
                           % comment out the preceeding one
                           % to use AE virtual fonts.
  \usepackage[francais]{babel}
  \begin{document}
  \showhyphens{signal container \'ev\'enement alg\`ebre}
  \showhyphens{signal container événement algèbre}
  \end{document}
  ```

- check the hyphenations proposed by TeX in your log-file; in French you should get with both 7-bit and 8-bit encodings
  `si-gnal contai-ner évé-ne-ment al-gèbre`.
  Do not care about how accented characters are displayed in the log-file, what matters is the position of the '`-`' hyphen signs *only*.

If they are all correct, your installation (probably) works fine, if one (or more) is (are) wrong, ask a local wizard to see what's going wrong and perform the test again (or e-mail me about what happens).

Frequent mismatches:

- you get `sig-nal con-tainer`, this probably means that the hyphenation patterns you are using are for US-English, not for French;

- you get no hyphen at all in `évé-ne-ment`, this probably means that you are using CM fonts and the macro `\accent` to produce accented characters. Using EC fonts with built-in accented characters or MlTeX with CM fonts avoids this type of mismatch.

`frenchb` has been improved using helpful suggestions from many people, the main contributions came from Vincent Jalby. Thanks to all of them!

## Changes

First version released: 1.1 as of 1996/05/31 part of babel-3.6beta.

**Changes in version 1.1b**: update for babel-3.6.

**Changes in version 1.2**: new command `\nombre` to format numbers; removed command `\fup` borrowed from the `french` package (`\up` does a better job in LaTeX 2$\varepsilon$); also removed aliases `\french` and `\english` (frenchb.cfg is a better place for these).

**Changes in version 1.3**:

- The 'xspace' package, when present, now controls spacing after all (sensible) macros, formerly 'xspace' worked on `\fg` (suggested by Vincent Jalby);
- spacing after opening and before closing guillemets improved as suggested by Thierry Bouche;
- a replacement for poor looking guillemets in OT1 encoding (*math* fonts are used to emulate them) is provided if the file `ot2wncyr.fd` is found: this file defines an OT2 encoding using AMS Cyrillic fonts; these have built-in guillemets, they are *text* fonts, are free, and are distributed as METAFONT and Type1 fonts (good for pdftex!), the replacement was suggested by Gérard Degrez; if the files `ot2wncyr.fd`, `wncy*.mf`, `wncy*.tfm`, and `wncy*.pfb` aren't available on your system, you *should* get them from CTAN; if your system complains about missing `wncy*.tfm` fonts, it means your TeX system is *incomplete*, as a quick fix, you can either remove `ot2wncyr.fd` or add the command `\LasyGuillemets` to the preamble of your document or to `frenchb.cfg`, then `frenchb` will behave as it did in the previous versions. In case Cyrillic guillemets do not fit, it is possible to pick French guillemets from any font using the command `\FrenchGuillemetsFrom` which requires 4 arguments `\FrenchGuillemetsFrom{`*coding*`}{`*font*`}{`*char-left*`}{`*char-right*`}`, for instance, add `\FrenchGuillemetsFrom{T1}{ppl}{19}{20}` to the preamble of your document or to `frenchb.cfg`, to pick French guillemets from Palatino (fontname=ppl, char 19 and 20 are left and right guillemets in T1-encoding). This was suggested by Michel Bovani.
- the environment 'itemize' has been redesigned in French according to specifications from Jacques André and Thierry Bouche; it is possible to switch back to the settings of version 1.2: add `\FrenchItemizeSpacingfalse` *after* loading `frenchb` in the preamble (this can be useful to process old documents);
- two switches have been added to go back to standard LaTeX list spacing `\FrenchItemizeSpacingfalse` and `\FrenchListSpacingfalse`;
- `\nombre` now properly handles signs in LaTeX 2$\varepsilon$;

- definition of `\dots` changed in French;
- in French, with the standard LaTeX classes, captions in figures and tables are printed with an endash instead of a colon.

**Changes in version 1.4**:

- the redefinition of`\@makecaption` is changed not to overwrite the changes made by some classes (koma-script, amsart, ua-thesis. . . ) as pointed out by Werner Lemberg;
- a hook, `\FrenchGuillemetsFrom`, is provided to pick French guillemets from any font (suggested by Michel Bovani, works *only* in OT1-encoding);
- a hook, `\FrenchLabelItem`, is provided to enable marks other than '–' (`\textendash`) in French lists (also suggested by Michel Bovani);
- `\DecimalMathComma`, `\StandardMathComma`, `\ThinSpaceInFrenchNumbers` added;
- `\FrenchLayout` and `\StandardLayout` added;
- an example of customization file `frenchb.cfg` is included in `frenchb.dtx`.

**Changes in version 1.5**:

- The settings for spacing in French lists are no longer tuned at the `\@trivlist` level but within `\list`; this enables the users to provide their own settings for the lists they define with `\list` (suggested by P. Pichaureau). Although the layout of the standard lists *has not changed*, some lists, based directly on `\@trivlist` or `\trivlist`, could possibly be typeset differently when upgrading from version 1.4 to 1.5. The command `\FrenchOldTrivlisttrue` could then, be useful to process older documents.
- Apart from the global hook `\FrenchLabelItem`, it is now also possible to change the 'labelitems' at any level (1 to 4) in French, using the standard LaTeX syntax, for instance: `\renewcommand{\labelitemii}{\textbullet}`.
- The internal name for the French language has been changed from `frenchb` to `french`, it means that `\captionsfrench`, `\datefrench`, `\extrasfrench`, `\noextrasfrench`, are to be used now instead of `\captionsfrenchb`, `\datefrenchb`, `\extrasfrenchb`, `\noextrasfrenchb`.
- From version 1.5f on, it is possible to customize the space inserted before a colon in French; its default value is `\space` (as in previous versions of `frenchb`), but it can now be changed to `\thinspace` by redefining `\Fcolonspace`.

**Changes in version 1.6**:

- Two new commands are provided: `\AddThinSpaceBeforeFootnotes` and `\FrenchFootnotes`; they enable to typeset footnotes according to the French IN's specifications (suggested by Jacques André).
- `frenchb` will make use of `\textdegree` (TS1 encoding) to typeset degrees whenever possible.
- Default guillemets have been changed *in OT1 encoding only* in version 1.6c: the former ones were picked up in the wncyr fonts, which forced the cyrillic encoding to OT2 when babel was loaded with option `frenchb` before a cyrillic language (`russian`, `ukrainian` or `bulgarian`). Now, LM fonts for T1-encoding are available, we pick up the guillemets from them, for OT1-encoding.
- The config file `frenchb.cfg` is now loaded only when LaTeX $2_\varepsilon$ is used (from version 1.6g).
- The command `\degres` now works also in math mode (from version 1.6g), but not in bold versions (it is a text character basically).

## Customizing frenchb: an example of configuration file

```
⟨*cfg⟩
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% If you want to customize frenchb, please DO NOT hack into the code,
%% copy this file into a directory searched by TeX, preferably a
%% personal one on multi-user systems, and customize it as you like.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% WARNING: If you exchange your documents with colleagues using
%% a different TeX installation, it is best NOT TO HAVE a frenchb.cfg
%% file, and add instead the customization commands to the preamble
%% of your documents after babel and frenchb have been loaded.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% Uncomment the following line if you dislike frenchb automatically
%% adding a space before French double punctuation (see frenchb.dtx
%% for more information).
%%
%%\NoAutoSpaceBeforeFDP
%%
%% frenchb inserts a normal space before a colon in French (following
%% I.N. specifications), but some people prefer a thinspace instead;
%% uncomment the following line if you prefer a thinspace.
%%
%%\renewcommand{\Fcolonspace}{\thinspace}
%%
%% Uncomment the following line to force lasy font emulation for
%% French guillemets in OT1 encoding; Polish guillemets are
%% FAR BETTER LOOKING, you should normally leave this line commented out.
%%
```

```
%%\LasyGuillemets
%%
%% The following option is for backward compatibility ONLY
%% (it might conflict with the russian option of babel).
%% Uncomment the following line if you want to get back to
%% the default guillemets used in versions $<$ 1.6b for OT1-encoding
%% (no effect in T1-encoding).
%%
%%\CyrillicGuillemets
%%
%% If you dislike Polish guillemets in OT1-encoding, you can easily
%% pick up yours from other fonts, examples below: Palatino or
%% cyrillic CMR T2A encoded font.
%% The four arguments are: encoding, font name, character codes of
%% the opening and closing guillemets in the selected font.
%%
%%\FrenchGuillemetsFrom{T1}{ppl}{19}{20}
%%\FrenchGuillemetsFrom{T2A}{cmr}{190}{191}
%%
%% Uncomment the following line if you don't want frenchb to change
%% LaTeX standard settings for vertical spacing in 'itemize' lists.
%% LaTeX standard settings were used by frenchb in versions < 1.3,
%% so you may want to uncomment the following line to process older
%% documents.
%%
%%\FrenchItemizeSpacingfalse
%%
%% frenchb tunes vertical spaces in all lists (at \list level),
%% uncomment the following line if you prefer LaTeX standard settings
%% (this flag is new in version 1.3). When \ifFrenchItemizeSpacing is
%% 'true' (default setting), this flag has no effect on itemize lists.
%%
%%\FrenchListSpacingfalse
%%
%% The command \FrenchLabelItem can be used to set the 'labelitems'
%% for all levels in French. For instance, to get long dashes, use:
%%
%%\renewcommand{\FrenchLabelItem}{\textemdash}
%%
%% You can also have different 'labelitems' for each level in French:
%%
%%\renewcommand{\Frlabelitemi}{\textemdash}
%%\renewcommand{\Frlabelitemii}{\bfseries\textendash}
%%\renewcommand{\Frlabelitemiii}{\textbullet}
%%\renewcommand{\Frlabelitemiv}{}
%%
%% Uncomment the following line if you want a thinspace to be added
%% in the running text before the footnote number or symbol.
%%
%%\AddThinSpaceBeforeFootnotes
```

```
%%
%% Uncomment the following line if you want footnotes to be typeset
%% according the specifications of the French ‘Imprimerie Nationale’.
%%
%%\FrenchFootnotes
%%
%% Uncomment the following line if you want the command \nombre to
%% insert thin spaces (1/6em) instead of normal spaces (0.3em)
%% between slices of 3 digits when typesetting numbers in French.
%%
%%\ThinSpaceInFrenchNumbers
%%
%% If you are bored to type {,} every time you want a decimal comma in
%% math mode, uncomment the following line (and don’t forget to add a
%% forced space after the comma in lists and intervals: $(x,\ y)$ and
%% $[0,\ 1]$). I suggest NOT TO UNCOMMENT the following line,
%% and to switch between the decimal comma and the standard comma
%% using \DecimalMathComma and \StandardMathComma inside the document.
%%
%%\DecimalMathComma
%%
%% If you do not want the appearance of lists to depend on the current
%% language, add ONE of these two commands: \FrenchLayout (for a
%% document mainly in French) or \StandardLayout (for a document
%% mainly in English).
%%
%%\FrenchLayout
%%\StandardLayout
%%
%% With the standard LaTeX classes (article.cls, report.cls, book.cls)
%% you can change the separator used in figure and table captions in
%% French with \CaptionSeparator, for instance (the default value of
%% \CaptionSeparator in French is ’\space\textendash\space’):
%%
%%\addto\captionsfrench{\def\CaptionSeparator{\space\textemdash\space}}
%%
%% You might want to change some of the translations of caption names,
%% you can do it this way, for instance:
%%
%%\addto\captionsfrench{\def\figurename{{\scshape Figure}}}
%%\addto\captionsfrench{\def\tablename{{\scshape Table}}}
%%\addto\captionsfrench{\def\proofname{Preuve}}
%%
%% If French guillemets are available on your keyboard, you can use
%% them instead of the commands \og and \fg: REPLACE << and >> in
%% the following code by your ready-made guillemets (2 occurrences
%% for each) AND uncomment the resulting code.
%% WARNINGS:
%% 1) This will reduce the portability of your source files!
%% 2) This adds two active characters.
%% 3) If you add this stuff to the preamble of a document, you'll
%%    need to wrap it in \makeatletter ... \makeatother.
%%
%% \ifx\mule@def\undefined
%%   \initiate@active@char{<<}
%%   \initiate@active@char{>>}
%%   \AtBeginDocument{%
%%     \def<<{\og\ignorespaces}%
%%     \def>>{{\fg}}%
%%     }%
%% \else %%% For the CJK package, see MULEenc.sty.
%%   \mule@def{11}{\og\ignorespaces}
%%   \mule@def{27}{{\fg}}
%% \fi
%%
%% The rest is for compatibility with Bernard Gaulle's french.sty:
%% if you want to typeset with babel/frenchb files originally written
%% for french.sty, you will probably want to uncomment these lines.
%%
%% Uncomment these lines if you want the aliases \numero and \Numero
%% (as in french.sty) for \no and \No.
%%
%%\let\numero=\no
%%\let\Numero=\No
%%
%% Uncomment the following lines if you want to substitute `\bsc' to
%% either (or both) of french.sty macros `\fsc' and `\lsc'.
%%
%%\let\fsc=\bsc
%%\let\lsc=\bsc
%%
%% Uncomment the following line if you want to substitute `\up' to
%% french.sty macro `\fup'.
%%
%%\let\fup=\up
%%
%% Uncomment the following lines if you intend to use  commands \french
%% and \english to switch between languages, as in french.sty.
%%
%%\def\french{\leavevmode\selectlanguage{french}}
%%\def\english{\leavevmode\selectlanguage{english}}
%%
%% Uncomment the following lines if you intend to use the command
%% \AllTeX to refer to all flavours of TeX, (this command has been
%% made popular by french.sty).
%%
%%\newcommand*{\AllTeX}{%
%%                (L\kern-.36em\raise.3ex\hbox{\sc a}\kern-.15em)%
%%                   T\kern-.1667em\lower.7ex\hbox{E}\kern-.125emX}
⟨/cfg⟩
```

# 29 TeXnical details

## 29.1 Initial setup

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
⟨*code⟩
%%
\LdfInit\CurrentOption\datefrench
```

`\ifLaTeX`
`\ifLaTeXe`

To check the format in use (plain, LaTeX or LaTeX 2$_\varepsilon$), we'll need two new 'if'.

```
\newif\ifLaTeX
\ifx\magnification\@undefined\LaTeXtrue\fi
\newif\ifLaTeXe
\ifx\@compatibilitytrue\@undefined\else\LaTeXetrue\fi
```

`\if@Two@E`

We will need another 'if' : `\if@Two@E` is true if and only if LaTeX 2$_\varepsilon$ is running *not* in compatibility mode. It is used in the definitions of the command `\nombre` and `\up`. The definition is somewhat complicated, due to the fact that `\if@compatibility` is not recognized as a `\if` in LaTeX-2.09 based formats.

```
\newif\if@Two@E \@Two@Etrue
\def\@FI@{\fi}
\ifx\@compatibilitytrue\@undefined
  \@Two@Efalse \def\@FI@{\relax}
\else
  \if@compatibility \@Two@Efalse \fi
\@FI@
```

Check if hyphenation patterns for the French language have been loaded in language.dat; we allow for the names 'french', 'francais', 'canadien' or 'acadian'. The latter two are both names used in Canada for variants of French that are in use in that country.

```
\ifx\l@french\@undefined
  \ifx\l@francais\@undefined
    \ifx\l@canadien\@undefined
      \ifx\l@acadian\@undefined
        \@nopatterns{French}
        \adddialect\l@french0
      \else
        \let\l@french\l@acadian
      \fi
    \else
      \let\l@french\l@canadien
    \fi
  \else
    \let\l@french\l@francais
  \fi
\fi
```

After the statement above `\l@french` is sure to always be defined.

The internal name for the French language will be 'french'; 'francais' and 'frenchb' will be synonymous for 'french': first let both names use the same hyphenation patterns. Later we will have to set aliases for `\captionsfrench`, `\datefrench`, `\extrasfrench` and `\noextrasfrench`. As French uses the standard values of `\lefthyphenmin` (2) and `\righthyphenmin` (3), no special setting is required here.

```
\def\CurrentOption{french}
\ifx\l@francais\@undefined
  \let\l@francais\l@french
\fi
\ifx\l@frenchb\@undefined
  \let\l@frenchb\l@french
\fi
```

When this language definition file was loaded for one of the Canadian versions of French we need to make sure that a suitable hyphenation pattern register will be found by TeX.

```
\ifx\l@canadien\@undefined
  \let\l@canadien\l@french
\fi
\ifx\l@acadian\@undefined
  \let\l@acadian\l@french
\fi
```

This language definition can be loaded for different variants of the French language. The 'key' babel macros are only defined once, using 'french' as the language name, but `frenchb` and `francais` are synonymous. As some users who choose `frenchb` or `francais` as option of babel, might customize `\captionsfrenchb` or `\captionsfrancais` in the preamble, we merge their changes at the `\begin{document}` when they do so (only possible in LaTeX $2_\varepsilon$). The other three commands for `\date`, `\extras`, and `\noextras` are not likely to be modified by end-users. The other variants of languages are defined by checking if the relevant option was used and then adding one extra level of expansion.

```
\def\datefrancais{\datefrench}
\def\extrasfrancais{\extrasfrench}
\def\noextrasfrancais{\noextrasfrench}
\def\datefrenchb{\datefrench}
\def\extrasfrenchb{\extrasfrench}
\def\noextrasfrenchb{\noextrasfrench}
\ifLaTeXe
  \AtBeginDocument{\let\captions@French\captionsfrench
                    \ifx\captionsfrenchb\@undefined
                       \let\captions@Frenchb\relax
                    \else
                       \let\captions@Frenchb\captionsfrenchb
                    \fi
                    \ifx\captionsfrancais\@undefined
```

```
                      \let\captions@Francais\relax
                   \else
                      \let\captions@Francais\captionsfrancais
                   \fi
                   \def\captionsfrench{\captions@French
                          \captions@Francais\captions@Frenchb}%
                   \def\captionsfrancais{\captionsfrench}%
                   \def\captionsfrenchb{\captionsfrench}%
                   \iflanguage{french}{\captionsfrench}{}%
                  }
\else
  \def\captionsfrancais{\captionsfrench}
  \def\captionsfrenchb{\captionsfrench}
\fi
\@ifpackagewith{babel}{canadien}{%
  \def\captionscanadien{\captionsfrench}%
  \def\datecanadien{\datefrench}%
  \def\extrascanadien{\extrasfrench}%
  \def\noextrascanadien{\noextrasfrench}%
  }{}
\@ifpackagewith{babel}{acadian}{%
  \def\captionsacadian{\captionsfrench}%
  \def\dateacadian{\datefrench}%
  \def\extrasacadian{\extrasfrench}%
  \def\noextrasacadian{\noextrasfrench}%
  }{}
```

`\extrasfrench`
`\noextrasfrench`

The macro `\extrasfrench` will perform all the extra definitions needed for the French language. The macro `\noextrasfrench` is used to cancel the actions of `\extrasfrench`.

In French "apostrophe" is used in hyphenation in expressions like `l'ambulance` (French patterns provide entries for this kind of words). This means that the `\lccode` of "apostrophe" has to be non null in French for proper hyphenation of those expressions, and to be reset to null when exiting French.

```
\@namedef{extras\CurrentOption}{\lccode`\'=`\'}
\@namedef{noextras\CurrentOption}{\lccode`\'=0}
```

It is best to use LaTeX 2$_\varepsilon$'s font changing commands, and to emulated those we need when they are not available, as in PlainTeX or LaTeX-2.09. Be aware that old commands `\sc`, `\it`, *etc.* exist in LaTeX 2$_\varepsilon$, but they behave like they did in LaTeX-2.09 (i. e., they switch back to `\normalfont` instead of keeping the other font attributes unchanged).

```
\ifx\scshape\@undefined
  \ifx\sc\@undefined
     \let\scshape\relax
  \else
     \let\scshape\sc
  \fi
\fi
```

```
\ifx\emph\@undefined
  \ifx\em\@undefined
     \let\emph\relax
  \else
     \def\emph#1{\em #1}
  \fi
\fi
```

## 29.2 Caption names

The next step consists of defining the French equivalents for the LaTeX caption names.

In French, captions in figures and tables should be printed with endash ('–') instead of the standard ':'. If another separator is preferred, see in the example configuration file above (section 28) how to change the default value of `\CaptionSeparator`.

We first store the standard definition of `\@makecaption` (e.g., the one provided in article.cls, report.cls, book.cls which is frozen for LaTeX 2$_\varepsilon$ according to Frank Mittelbach), in `\STD@makecaption`. 'ATBeginDocument' we compare it to its current definition (some classes like koma-script classes, AMS classes, ua-thesis.cls... change it). If they are identical, frenchb just adds a hook called `\CaptionSeparator` to `\@makecaption`, `\CaptionSeparator` defaults to ': ' as in the standard `\@makecaption`, and will be changed to ' – ' in French. If the definitions differ, frenchb doesn't overwrite the changes, but prints a message in the .log file.

```
\ifLaTeXe
  \long\def\STD@makecaption#1#2{%
    \vskip\abovecaptionskip
    \sbox\@tempboxa{#1: #2}%
    \ifdim \wd\@tempboxa >\hsize
      #1: #2\par
    \else
      \global \@minipagefalse
      \hb@xt@\hsize{\hfil\box\@tempboxa\hfil}%
    \fi
    \vskip\belowcaptionskip}
  \def\CaptionSeparator{\string:\space}
  \long\def\FB@makecaption#1#2{%
    \vskip\abovecaptionskip
    \sbox\@tempboxa{#1\CaptionSeparator #2}%
    \ifdim \wd\@tempboxa >\hsize
      #1\CaptionSeparator #2\par
    \else
      \global \@minipagefalse
      \hb@xt@\hsize{\hfil\box\@tempboxa\hfil}%
    \fi
    \vskip\belowcaptionskip}
  \AtBeginDocument{%
  \ifx\@makecaption\STD@makecaption
      \let\@makecaption\FB@makecaption
  \else
    \ifx\@makecaption\@undefined\else
      \PackageWarning{frenchb.ldf}%
        {The definition of \protect\@makecaption\space
         has been changed,\MessageBreak
         frenchb will NOT customize it;\MessageBreak reported}%
    \fi
  \fi
  \let\FB@makecaption\relax
  \let\STD@makecaption\relax}
  \expandafter\addto\csname noextras\CurrentOption\endcsname{%
     \def\CaptionSeparator{\string:\space}}
\fi
```

`\captionsfrench` The macro `\captionsfrench` defines all strings used in the four standard document classes provided with LaTeX. Some authors do not like some of these names; it is easy to change them in the preamble *after* loading `frenchb` (or in your file `frenchb.cfg`), e.g `\addto\captionsfrench{\def\figurename{Figure}}` will print 'Figure' in Roman instead of 'Fig.'.

```
\ifLaTeX
\@namedef{captions\CurrentOption}{%
   \def\refname{R\'ef\'erences}%
   \def\abstractname{R\'esum\'e}%
   \def\bibname{Bibliographie}%
   \def\prefacename{Pr\'eface}%
   \def\chaptername{Chapitre}%
   \def\appendixname{Annexe}%
   \def\contentsname{Table des mati\`eres}%
   \def\listfigurename{Table des figures}%
   \def\listtablename{Liste des tableaux}%
   \def\indexname{Index}%
   \def\figurename{{\scshape Fig.}}%
   \def\tablename{{\scshape Tab.}}%
   \def\CaptionSeparator{\space\textendash\space}%
```

"Première partie" instead of "Part I"

```
   \def\partname{\protect\@Fpt partie}%
   \def\@Fpt{{\ifcase\value{part}\or Premi\`ere\or Deuxi\`eme\or
   Troisi\`eme\or Quatri\`eme\or Cinqui\`eme\or Sixi\`eme\or
   Septi\`eme\or Huiti\`eme\or Neuvi\`eme\or Dixi\`eme\or Onzi\`eme\or
   Douzi\`eme\or Treizi\`eme\or Quatorzi\`eme\or Quinzi\`eme\or
   Seizi\`eme\or Dix-septi\`eme\or Dix-huiti\`eme\or Dix-neuvi\`eme\or
   Vingti\`eme\fi}\space\def\thepart{}}%
   \def\pagename{page}%
   \def\seename{{\emph{voir}}}%
   \def\alsoname{{\emph{voir aussi}}}%
   \def\enclname{P.~J. }%
```

```
  \def\ccname{Copie \`a }%
  \def\headtoname{}%
  \def\proofname{D\'emonstration}% for AMS-\LaTeX
  \def\glossaryname{Glossaire}%
  }
\fi
```

## 29.3 Punctuation

The 'double punctuation' characters (`;` `!` `?` and `:`) have to be made `\active` for an automatic control of the amount of space to insert before them.

```
\initiate@active@char{:}
\initiate@active@char{;}
\initiate@active@char{!}
\initiate@active@char{?}
```

We specify that the French group of shorthands should be used.

```
\expandafter\addto\csname extras\CurrentOption\endcsname{%
 \languageshorthands{french}}
```

These characters are 'turned on' once, later their definition may vary.

```
\expandafter\addto\csname extras\CurrentOption\endcsname{%
 \bbl@activate{:}\bbl@activate{;}%
 \bbl@activate{!}\bbl@activate{?}}
\expandafter\addto\csname noextras\CurrentOption\endcsname{%
 \bbl@deactivate{:}\bbl@deactivate{;}%
 \bbl@deactivate{!}\bbl@deactivate{?}}
```

One more thing `\extrasfrench` needs to do is to make sure that `\frenchspacing` is in effect. If this is not the case the execution of `\noextrasfrench` will switch it off again.

```
\expandafter\addto\csname extras\CurrentOption\endcsname{%
 \bbl@frenchspacing}
\expandafter\addto\csname noextras\CurrentOption\endcsname{%
 \bbl@nonfrenchspacing}
```

`\french@sh@;@` We have to tune the amount of white space before `;` `!` `?` and `:`. This should only happen in horizontal mode, hence the test `\ifhmode`.

In horizontal mode, if a space has been typed before ';' we remove it and put an unbreakable `\thinspace` instead. If no space has been typed, we add `\FDP@thinspace` which will be defined, up to the user's wishes, as an automatic added thin space, or as `\@empty`.

```
\declare@shorthand{french}{;}{%
   \ifhmode
     \ifdim\lastskip>\z@
       \unskip\penalty\@M\thinspace
     \else
       \FDP@thinspace
     \fi
   \fi
```

Now we can insert a ; character.

```
    \string;}
```

`\french@sh@!@` `\french@sh@?@` Because these definitions are very similar only one is displayed in a way that the definition can be easily checked.

```
\declare@shorthand{french}{!}{%
    \ifhmode
      \ifdim\lastskip>\z@
        \unskip\penalty\@M\thinspace
      \else
        \FDP@thinspace
      \fi
    \fi
    \string!}
```

```
\declare@shorthand{french}{?}{%
    \ifhmode
      \ifdim\lastskip>\z@
        \unskip\penalty\@M\thinspace
      \else
        \FDP@thinspace
      \fi
    \fi
    \string?}
```

`\french@sh@:@` The ':' requires a normal space before it, instead of a `\thinspace`.

```
\declare@shorthand{french}{:}{%
    \ifhmode
      \ifdim\lastskip>\z@
        \unskip\penalty\@M\Fcolonspace
      \else
        \FDP@colonspace
      \fi
    \fi
    \string:}
```

`\Fcolonspace` `\AutoSpaceBeforeFDP` `\NoAutoSpaceBeforeFDP` `\FDP@thinspace` and `\FDP@space` are defined as unbreakable spaces by `\AutoSpaceBeforeFDP` or as `\@empty` by `\NoAutoSpaceBeforeFDP`. Default is `\AutoSpaceBeforeFDP`.

```
\newcommand{\Fcolonspace}{\space}
\def\AutoSpaceBeforeFDP{%
      \def\FDP@thinspace{\penalty\@M\thinspace}%
      \def\FDP@colonspace{\penalty\@M\Fcolonspace}}
\def\NoAutoSpaceBeforeFDP{\let\FDP@thinspace\@empty
                          \let\FDP@colonspace\@empty}
\AutoSpaceBeforeFDP
```

\system@sh@:@
\system@sh@!@
\system@sh@?@
\system@sh@;@

When the active characters appear in an environment where their French behaviour is not wanted they should give an 'expected' result. Therefore we define shorthands at system level as well.

```
\declare@shorthand{system}{:}{\string:}
\declare@shorthand{system}{!}{\string!}
\declare@shorthand{system}{?}{\string?}
\declare@shorthand{system}{;}{\string;}
```

## 29.4 French quotation marks

EC/LM fonts and standard PostScript fonts have built-in guillemets, we of course use them. CM fonts have no French guillemets built-in, so we have to emulate them:

- if a file 't1lmr.fd' is found, we expect the LM fonts to be present on the system, so we pick up French guillemets from them; before version 1.6b, French guillemets were borrowed from the 'wncyr' fonts, this has been changed because it resulted in a conflict with the russian option of babel. Former Cyrillic guillemets are still available through the command `\CyrillicGuillemets` (for backward compatibility).
- otherwise we use math symbols, either LaTeX's 'lasy' font if available, or TeX symbols `\ll` and `\gg` otherwise;

The standard names for French quotation marks are `\guillemotleft` and `\guillemotright`, these commands are defined in `t1enc.def` for T1-encoding. In LaTeX 2$_\varepsilon$ formats, we first define these commands for OT1-encoding (for CM fonts), using either the Polish 'pl*' fonts or LaTeX's 'lasy' fonts. Other local text encodings *should* define them too, but if they don't, the OT1 definitions will be used. The standard PostScript fonts should *always* be used together with T1-encoding (*not* OT1), for proper hyphenation and built-in (better looking) French quotation marks.

The next step is to provide correct spacing after `\guillemotleft` and before `\guillemotright` : a space precedes and follows quotation marks but no line break is allowed neither *after* the opening one, nor *before* the closing one. `\guill@spacing` which does the spacing, has been fine tuned by Thierry Bouche.

The top macros for quotation marks will be called `\og` ("$\underline{\text{o}}$uvrez $\underline{\text{g}}$uillemets") and `\fg` ("$\underline{\text{f}}$ermez $\underline{\text{g}}$uillemets").

The top level definitions for French quotation marks are switched on and off through the `\extrasfrench` `\noextrasfrench` mechanism. Outside French, `\og` and `\fg` will typeset standard English opening and closing double quotes.

As `\DeclareTextCommand` cannot be used after the `\begin{document}` we introduce internal definitions `\begin@guill` and `\end@guill`.

We'll try to be smart to users of D. Carlisle's `xspace` package: if this package is loaded there will be no need for `{}` or `\ ` to get a space after `\fg`.

\guillemotleft
\guillemotright
\og
\fg

In LaTeX 2ε we provide a dummy definition for \og and \fg, just to display an error message in case \og or \fg have been defined elsewhere.

```
\newcommand{\og}{\@empty}
\newcommand{\fg}{\@empty}
```

\FrenchGuillemetsFrom

```
\ifLaTeXe
  \def\FrenchGuillemetsFrom#1#2#3#4{%
    \DeclareFontEncoding{#1}{}{}%
    \DeclareFontSubstitution{#1}{#2}{m}{n}%
    \DeclareTextCommand{\guillemotleft}{OT1}{%
      {\fontencoding{#1}\fontfamily{#2}\selectfont\char#3}}%
    \DeclareTextCommand{\guillemotright}{OT1}{%
      {\fontencoding{#1}\fontfamily{#2}\selectfont\char#4}}}
```

\CyrillicGuillemets
\PolishGuillemets
\LasyGuillemets
\bbl@frenchguillemets
\bbl@nonfrenchguillemets

```
  \def\CyrillicGuillemets{\FrenchGuillemetsFrom{OT2}{wncyr}{60}{62}}
  \def\PolishGuillemets{\FrenchGuillemetsFrom{T1}{lmr}{19}{20}}
  \def\LasyGuillemets{%
    \DeclareTextCommand{\guillemotleft}{OT1}{\hbox{%
      \fontencoding{U}\fontfamily{lasy}\selectfont(\kern-0.20em(}}%
    \DeclareTextCommand{\guillemotright}{OT1}{\hbox{%
      \fontencoding{U}\fontfamily{lasy}\selectfont)\kern-0.20em)}}}
  \IfFileExists{t1lmr.fd}{\PolishGuillemets}{\LasyGuillemets}
  \DeclareTextSymbolDefault{\guillemotleft}{OT1}
  \DeclareTextSymbolDefault{\guillemotright}{OT1}
  \def\guill@spacing{\penalty\@M\hskip.8\fontdimen2\font
                                plus.3\fontdimen3\font
                                minus.8\fontdimen4\font}
  \DeclareRobustCommand*{\begin@guill}{\leavevmode
                              \guillemotleft\penalty\@M\guill@spacing}
  \DeclareRobustCommand*{\end@guill}{\ifdim\lastskip>\z@\unskip\fi
                     \penalty\@M\guill@spacing\guillemotright\xspace}
  \AtBeginDocument{\ifx\xspace\@undefined\let\xspace\relax\fi}
  \def\bbl@frenchguillemets{\renewcommand{\og}{\begin@guill}%
                              \renewcommand{\fg}{\end@guill}}
  \def\bbl@nonfrenchguillemets{\renewcommand{\og}{``}%
           \renewcommand{\fg}{\ifdim\lastskip>\z@\unskip\fi ''}}
```

For *Plain*TeX, and LaTeX-2.09 we define \begin@guill and \end@guill using math symbols \ll and \gg.

```
\else
  \def\begin@guill{\leavevmode\raise0.25ex
                   \hbox{$\scriptscriptstyle\ll$}%
                   \penalty\@M\hskip.8\fontdimen2\font
                                plus.3\fontdimen3\font
                                minus.3\fontdimen4\font}
  \def\end@guill{\ifdim\lastskip>\z@\unskip\penalty\@M\fi
                 \penalty\@M\hskip.8\fontdimen2\font
```

```
                   plus.3\fontdimen3\font minus.3\fontdimen4\font
                   \raise0.25ex\hbox{$\scriptscriptstyle\gg$}}
  \let\xspace\relax
  \def\bbl@frenchguillemets{\let\og\begin@guill
                            \let\fg\end@guill}
  \def\bbl@nonfrenchguillemets{\def\og{``}%
                       \def\fg{\ifdim\lastskip>\z@\unskip\fi ''}}
\fi
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \bbl@frenchguillemets}
\expandafter\addto\csname noextras\CurrentOption\endcsname{%
  \bbl@nonfrenchguillemets}
```

## 29.5 French lists

`\bbl@frenchlistspacing`
`\bbl@nonfrenchlistspacing`

Vertical spacing in general lists should be shorter in French texts than the defaults provided by LaTeX. Note that the easy way, just changing values of vertical spacing parameters when entering French and restoring them to their defaults on exit would not work; as most lists are based on `\list` we will redefine `\list` (up to version 1.4, the more internal command `\@trivlist` was redefined, this ensured that all lists would have common settings, but didn't allow the users to provide their own settings, as pointed out by P. Pichaureau). If standard LaTeX settings are preferred, it is easy to issue the command `\FrenchListSpacingfalse` in the preamble or in `frenchb.cfg`. Please note that changing the flag `\FrenchListSpacing` will *not* take effect immediately, but next time language French is switched on.

The amount of vertical space before and after a list is given by `\topsep` + `\parskip` (+ `\partopsep` if the list starts a new paragraph). IMHO, `\parskip` should be added *only* when the list starts a new paragraph, so I subtract `\parskip` from `\topsep` and add it back to `\partopsep`; this will normally make no difference because `\parskip`'s default value is 0pt, but will be noticeable when `\parskip` is *not* null.

Of course, this code is only for LaTeX.

```
\newif\ifFrenchListSpacing \FrenchListSpacingtrue
\newif\ifFrenchOldTrivlist
\ifLaTeX
  \let\listORI\list
```

`\endlist` is not redefined, but `\endlistORI` is provided for the users who prefer to define their own lists from the original command, they can code: `\begin{listORI}{}{} \end{listORI}`.

```
  \let\endlistORI\endlist
  \def\FR@listsettings{%
    \setlength{\itemsep}{0.4ex plus 0.2ex minus 0.2ex}%
    \setlength{\parsep}{0.4ex plus 0.2ex minus 0.2ex}%
    \setlength{\topsep}{0.8ex plus 0.4ex minus 0.4ex}%
    \setlength{\partopsep}{0.4ex plus 0.2ex minus 0.2ex}%
```

`\parskip` is of type 'skip', its mean value only (*not the glue*) should be substracted from `\topsep` and added to `\partopsep`, so convert `\parskip` to a 'dimen' using `\@tempdima`.

```
    \@tempdima=\parskip
    \addtolength{\topsep}{-\@tempdima}%
    \addtolength{\partopsep}{\@tempdima}}%
  \def\listFR#1#2{\listORI{#1}{\FR@listsettings #2}}%
```

Keep the `\@trivlist` redefinition from version 1.4 (only for compatibility reasons). The flag `\ifFrenchOldTrivlist` could happen to be useful to process older documents.

```
  \let\@trivlistORI\@trivlist
  \def\@trivlistFR{\FR@listsettings\@trivlistORI}
  \def\bbl@frenchlistspacing{\ifFrenchListSpacing\let\list\listFR\fi
                     \ifFrenchOldTrivlist\let\@trivlist\@trivlistFR
                                          \let\list\listORI\fi}
  \def\bbl@nonfrenchlistspacing{\let\list\listORI
                                    \let\@trivlist\@trivlistORI}
  \expandafter\addto\csname extras\CurrentOption\endcsname{%
    \bbl@frenchlistspacing}
  \expandafter\addto\csname noextras\CurrentOption\endcsname{%
    \bbl@nonfrenchlistspacing}
\fi
```

`\bbl@frenchitemize`
`\bbl@nonfrenchitemize`

The layout of French itemize-lists has changed between version 1.2 and 1.3. Jacques André and Thierry Bouche pointed out to me that vertical spacing between items, before and after the list, should *null* with *no glue* added. Moreover horizontal spacing was not correct either: the item labels of a first level list should be vertically aligned on the paragraph's first character (at `\parindent` from the left margin). Checking the book "Lexique des règles typographiques en usage à l'Imprimerie nationale" confirmed their points, so the itemize environment has been totally redefined for French in version 1.3.

Several people pointed out to me that the layout of lists can either be set by the current language or be tuned independently of the language as part of the layout of the document. So hooks should be provided to switch off the special settings made in `frenchb`. Setting `\FrenchItemizeSpacingfalse` *after* loading `frenchb` in the preamble switches back to settings of version 1.2 (this can be useful to process old documents). If you want all list spacings to be exactly what they are in standard LaTeX $2_\varepsilon$, then add also `\FrenchListSpacingfalse` in the preamble *after* loading `frenchb`. Both switches can also be set in `frenchb.cfg`. Please note that changing the flags `\FrenchItemizeSpacing` and `\FrenchListSpacing` will *not* take effect immediately, but next time language French is switched on.

The • is never used in French itemize-lists, a long dash '–' is preferred for all levels. The item label used in French is stored in `\FrenchLabelItem}`, it defaults to '–' and can be changed using `\renewcommand` (see the example configuration file above).

```
\newif\ifFrenchItemizeSpacing \FrenchItemizeSpacingtrue
\ifLaTeX
```

```
\newcommand{\FrenchLabelItem}{\textendash}
\newcommand{\Frlabelitemi}{\FrenchLabelItem}
\newcommand{\Frlabelitemii}{\FrenchLabelItem}
\newcommand{\Frlabelitemiii}{\FrenchLabelItem}
\newcommand{\Frlabelitemiv}{\FrenchLabelItem}
\def\bbl@frenchitemize{%
  \let\@ltiORI\labelitemi
  \let\@ltiiORI\labelitemii
  \let\@ltiiiORI\labelitemiii
  \let\@ltivORI\labelitemiv
  \let\itemizeORI\itemize
  \let\labelitemi\Frlabelitemi
  \let\labelitemii\Frlabelitemii
  \let\labelitemiii\Frlabelitemiii
  \let\labelitemiv\Frlabelitemiv
  \ifFrenchItemizeSpacing
```

Make sure `\itemize` uses `\@trivlistORI` even with `\FrenchOldTrivlisttrue`.

```
    \def\itemize{\let\@trivlist\@trivlistORI
      \ifnum \@itemdepth >\thr@@\@toodeep\else
      \advance\@itemdepth\@ne
      \edef\@itemitem{labelitem\romannumeral\the\@itemdepth}%
      \expandafter
      \listORI
      \csname\@itemitem\endcsname
      {\settowidth{\labelwidth}{\csname\@itemitem\endcsname}%
       \setlength{\leftmargin}{\labelwidth}%
       \addtolength{\leftmargin}{\labelsep}%
       \ifnum\@listdepth=0
         \setlength{\itemindent}{\parindent}%
       \else
         \addtolength{\leftmargin}{\parindent}%
       \fi
       \setlength{\itemsep}{\z@}%
       \setlength{\parsep}{\z@}%
       \setlength{\topsep}{\z@}%
       \setlength{\partopsep}{\z@}%
```

`\parskip` is of type 'skip', its mean value only (*not the glue*) should be substracted from `\topsep` and added to `\partopsep`, so convert `\parskip` to a 'dimen' using `\@tempdima`.

```
       \@tempdima=\parskip
       \addtolength{\topsep}{-\@tempdima}%
       \addtolength{\partopsep}{\@tempdima}}%
      \fi}%
   \fi}
```

The user's changes in labelitems are saved when leaving French for further use when switching back to French.

```
\def\bbl@nonfrenchitemize{\let\Frlabelitemi\labelitemi
                          \let\Frlabelitemii\labelitemii
```

```
                           \let\Frlabelitemiii\labelitemiii
                           \let\Frlabelitemiv\labelitemiv
                           \let\labelitemi\@ltiORI
                           \let\labelitemii\@ltiiORI
                           \let\labelitemiii\@ltiiiORI
                           \let\labelitemiv\@ltivORI
                           \let\itemize\itemizeORI}
  \expandafter\addto\csname extras\CurrentOption\endcsname{%
    \bbl@frenchitemize}
  \expandafter\addto\csname noextras\CurrentOption\endcsname{%
    \bbl@nonfrenchitemize}
\fi
```

## 29.6 French indentation of sections

`\bbl@frenchindent`
`\bbl@nonfrenchindent`

In French the first paragraph of each section should be indented, this is another difference with US-English. Add this code only in LaTeX.

```
\ifLaTeX
  \let\@aifORI\@afterindentfalse
  \def\bbl@frenchindent{\let\@afterindentfalse\@afterindenttrue
                       \@afterindenttrue}
  \def\bbl@nonfrenchindent{\let\@afterindentfalse\@aifORI
                         \@afterindentfalse}
  \expandafter\addto\csname extras\CurrentOption\endcsname{%
    \bbl@frenchindent}
  \expandafter\addto\csname noextras\CurrentOption\endcsname{%
    \bbl@nonfrenchindent}
\fi
```

## 29.7 Formatting footnotes

`\AddThinSpaceBeforeFootnotes`
`\FrenchFootnotes`
`\StandardFootnotes`

`\AddThinSpaceBeforeFootnotes` redefines `\@footnotemark` (whose definition is saved at the `\begin{document}` in order to include any customization that packages might have done) to add a thinspace before the number or symbol calling a footnote (any space typed in is removed first). `\AddThinSpaceBeforeFootnotes` has no effect on the layout of the footnote itself. Note that `\thanks` in `\maketitle` also relies on `\@footnotemark`, so the thinspace is added there too.

```
\newif\ifthinspace@FN
\newif\ifFR@fntlayout
\newdimen\parindentFFN
\ifLaTeXe
  \newcommand{\AddThinSpaceBeforeFootnotes}{\thinspace@FNtrue}
  \AtBeginDocument{\let\@footnotemarkORI\@footnotemark
                  \def\@footnotemarkFR{\leavevmode\unskip\unkern
                                      \,\@footnotemarkORI}%
                  \ifthinspace@FN
                    \let\@footnotemark\@footnotemarkFR
                  \fi}
```

We then define `\@makefntextFR`, a variant of `\@makefntext` which is responsible for the layout of footnotes, to match the specifications of the French 'Imprimerie Nationale': Footnotes will be indented by `\parindentFFN`, numbers (if any) typeset on the baseline (instead of textsuperscripts) and followed by a dot and an half quad space. Whenever symbols are used to number footnotes (as in `\thanks` for instance), we switch back to the standard layout (the French layout of footnotes is meant for footnotes numbered by arabic or roman digits). The value of `\parindentFFN` will be defined at the `\begin{document}`, as the maximum of `\parindent` and 1.5em.

```
  \def\ftnISsymbol{\@fnsymbol\c@footnote}
  \long\def\@makefntextFR#1{\ifx\thefootnote\ftnISsymbol
                               \@makefntextORI{#1}%
                             \else
                               \parindent=\parindentFFN
                               \rule\z@\footnotesep
                               \setbox\@tempboxa\hbox{\@thefnmark}%
                               \ifdim\wd\@tempboxa>\z@
                                 \llap{\@thefnmark}.\kern.5em
                               \fi #1
                             \fi}%
```

We now define two commands `\FrenchFootnotes` and `\StandardFootnotes` which select French or Standard layout for footnotes; we have to save the standard definition of `\@makefntext` at the `\begin{document}`, and then redefine `\@makefntext` according to the value of an internal flag set by `\FrenchFootnotes` and reset by `\StandardFootnotes`.

```
  \AtBeginDocument{\parindentFFN=\parindent
                   \ifdim\parindentFFN<1.5em\parindentFFN=1.5em\fi
                   \let\@makefntextORI\@makefntext
                   \long\def\@makefntext#1{%
                      \ifFR@fntlayout
                         \@makefntextFR{#1}%
                      \else
                         \@makefntextORI{#1}%
                      \fi
                         }%
                  }
  \newcommand{\FrenchFootnotes}{\FR@fntlayouttrue}
  \newcommand{\StandardFootnotes}{\FR@fntlayoutfalse}
\fi
```

## 29.8 Formatting numbers

`\DecimalMathComma` `\StandardMathComma`

As mentioned in the TEXbook p. 134, the comma is of type `\mathpunct` in math mode: it is automatically followed by a space. This is convenient in lists and intervals but unpleasant when the comma is used as a decimal separator in French: it has to be entered as `{,}`. `\DecimalMathComma` makes the comma be an ordinary character (of type `\mathord`) in French *only* (no space added); `\StandardMathComma` switches back to the standard behaviour of the comma.

```
\newcount\std@mcc
\newcount\dec@mcc
\std@mcc=\mathcode`\,
\dec@mcc=\std@mcc
\@tempcnta=\std@mcc
\divide\@tempcnta by "1000
\multiply\@tempcnta by "1000
\advance\dec@mcc by -\@tempcnta
\newcommand{\DecimalMathComma}{\iflanguage{french}%
                                  {\mathcode`\,=\dec@mcc}{}%
               \addto\extrasfrench{\mathcode`\,=\dec@mcc}}
\newcommand{\StandardMathComma}{\mathcode`\,=\std@mcc
             \addto\extrasfrench{\mathcode`\,=\std@mcc}}
\expandafter\addto\csname noextras\CurrentOption\endcsname{%
   \mathcode`\,=\std@mcc}
```

In English the decimal part starts with a point and thousands should be separated by a comma: an approximation of $1000\pi$ should be inputed as `$3{,}141.592{,}653$` in math mode and as `3,141.592,653` in text.

In French the decimal part starts with a comma and thousands should be separated by a space; the same approximation of $1000\pi$ should be inputed as `$3\;141{,}592\;653$` in math mode and as something like `3~141,592~653` in text: in math mode, the comma is of type `\mathpunct` (thus normally followed by a space) while the point is of type `\mathord` (no space added).

Thierry Bouche suggested that a second type of comma, of type `\mathord` would be useful in math mode, and proposed to introduce a command (named `\decimalsep` in this package), the expansion of which would depend on the current language.

Vincent Jalby suggested a command `\nombre` to conveniently typeset numbers: inputting `\nombre{3141,592653}` either in text or in math mode will format this number properly according to the current language (French or non-French).

`\nombre` accepts an optional argument which happens to be useful with the package 'dcolumn', it specifies the decimal separator used in the *source code*:

```
\newcolumntype{d}{D{,}{\decimalsep}{-1}}
   \begin{tabular}{d}\hline
     3,14 \\
     \nombre[,]{123,4567} \\
     \nombre[,]{9876,543}\\\hline
   \end{tabular}
```

will print a column of numbers aligned on the decimal point (comma or point depending on the current language), each slice of 3 digits being separated by a space or a comma according to the current language.

`\decimalsep` `\thousandsep` We need a internal definition, valid in both text and math mode, for the comma (`\@comma@`) and another one for the unbreakable fixed length space (no glue) used in French (`\f@thousandsep`).

The commands `\decimalsep` and `\thousandsep` get default definitions (for the English language) when `frenchb` is loaded; these definitions will be updated when the current language is switched to or from French.

```
\mathchardef\m@comma=\dec@mcc
\def\@comma@{\ifmmode\m@comma\else,\fi}
\def\f@thousandsep{\ifmmode\mskip5.5mu\else\penalty\@M\kern.3em\fi}
\def\ThinSpaceInFrenchNumbers{\def\f@thousandsep{%
                  \ifmmode\mskip3mu\else\penalty\@M\kern.16667em\fi}}
\newcommand{\decimalsep}{.}
\newcommand{\thousandsep}{\@comma@}
\expandafter\addto\csname extras\CurrentOption\endcsname{%
            \def\decimalsep{\@comma@}%
            \def\thousandsep{\f@thousandsep}}
\expandafter\addto\csname noextras\CurrentOption\endcsname{%
            \def\decimalsep{.}%
            \def\thousandsep{\@comma@}}
```

Signs can now be entered inside `\nombre`. When `\nombre` is used in text-mode, signs should be text symbols to get the series, shape... from the current text-font. When signs are not available in text-mode, we provide some defaults.

```
\providecommand{\textminus}{\textendash}
\providecommand{\textplusminus}{\ensuremath{\pm}}
\providecommand{\textminusplus}{\ensuremath{\mp}}
\def\fb@minus{\ifmmode-\else\textminus\fi}
\def\fb@plusminus{\ifmmode\pm\else\textplusminus\fi}
\def\fb@minusplus{\ifmmode\mp\else\textminusplus\fi}
```

`\nombre` The decimal separator used when *inputing* a number with `\nombre` *has to be a comma.* `\nombre` splits the inputed number into two parts: what comes before the first comma will be formatted by `\@integerpart` while the rest (if not empty) will be formatted by `\@decimalpart`. Both parts, once formatted separately will be merged together with between them, either the decimal separator `\decimalsep` or (in LaTeX 2ε *only*) the optional argument of `\nombre`.

```
\if@Two@E
  \newcommand{\nombre}[2][\decimalsep]{\def\@decimalsep{#1}%
          \@@nombre#2\@empty,\@empty,\@nil}
\else
  \def\@decimalsep{\decimalsep}
  \newcommand{\nombre}[1]{\@nombre#1\@empty,\@empty,\@nil}
\fi
\def\@firstofmany#1#2,{#1}
\def\@@nombre#1,#2,#3\@nil{%
        \def\nb@sign{}%
        \edef\nb@first{\@firstofmany #1\@empty,}%
        \edef\nb@suite{\@secondoftwo #1\@empty,}%
        \if+\nb@first   \def\nb@sign{+}\fi
        \if-\nb@first   \def\nb@sign{\fb@minus}\fi
        \expandafter\ifx\nb@first\pm \def\nb@sign{\fb@plusminus}\fi
```

```
      \expandafter\ifx\nb@first\mp \def\nb@sign{\fb@minusplus}\fi
      \ifx\@empty\nb@sign
        \let\@tmp\nb@suite\edef\nb@suite{\nb@first\@tmp}%
      \fi
    \nb@sign\expandafter\@nombre\nb@suite#2,#3\@nil}
\def\@nombre#1,#2,#3\@nil{%
      \ifx\@empty#2%
        \@integerpart{#1}%
      \else
        \@integerpart{#1}\@decimalsep\@decimalpart{#2}%
      \fi}
```

The easiest bit is the decimal part: We attempt to read the first four digits of the decimal part, if it has less than 4 digits, we just have to print them, otherwise `\thousandsep` has to be appended after the third digit, and the algorithm is applied recursively to the rest of the decimal part.

```
\def\@decimalpart#1{\@@decimalpart#1\@empty\@empty\@empty}
\def\@@decimalpart#1#2#3#4{#1#2#3%
  \ifx\@empty#4%
  \else
    \thousandsep\expandafter\@@decimalpart\expandafter#4%
  \fi}
```

Formatting the integer part is more difficult because the slices of 3 digits start from the *bottom* while the number is read from the top! This (tricky) code is borrowed from David Carlisle's comma.sty.

```
\def\@integerpart#1{\@@integerpart{}#1\@empty\@empty\@empty}
\def\@@integerpart#1#2#3#4{%
  \ifx\@empty#2%
    \@addthousandsep#1\relax
  \else
    \ifx\@empty#3%
      \@addthousandsep\@empty\@empty#1#2\relax
    \else
      \ifx\@empty#4%
        \@addthousandsep\@empty#1#2#3\relax
      \else
        \@@integerpartafterfi{#1#2#3#4}%
      \fi
    \fi
  \fi}
\def\@@integerpartafterfi#1\fi\fi\fi{\fi\fi\fi\@@integerpart{#1}}
\def\@addthousandsep#1#2#3#4{#1#2#3%
  \if#4\relax
  \else
    \thousandsep\expandafter\@addthousandsep\expandafter#4%
  \fi}
```

## 29.9 Dots…

LaTeX 2ε's standard definition of `\dots` in text-mode is `\textellipsis` which includes a `\kern` at the end; this space is not wanted in some cases (before a closing brace for instance) and `\kern` breaks hyphenation of the next word. We define `\Frtextellipsis` for French (in LaTeX 2ε only).

The `\if` construction in the LaTeX 2ε definition of `\dots` doesn't allow the use of `xspace` (`xspace` is always followed by a `\fi`), so we use the AMS-LaTeX construction of `\dots`; this has to be done 'AtBeginDocument' not to be overwritten when `amsmath.sty` is loaded after babel.

`\Frtextellipsis`

```
\ifLaTeXe
  \DeclareTextCommandDefault{\Frtextellipsis}{%
    .\kern\fontdimen3\font.\kern\fontdimen3\font.\xspace}
```

`\Mdots@` and `\Tdots@ORI` hold the definitions of `\dots` in Math and Text mode. They default to those of amsmath-2.0, and will revert to standard LaTeX definitions 'AtBeginDocument', if amsmath has not been loaded. `\Mdots@` doesn't change when switching from/to French, while `\Tdots@` is `\Frtextellipsis` in French and `\Tdots@ORI` otherwise.

```
  \newcommand{\Tdots@ORI}{\@xp\textellipsis}
  \newcommand{\Tdots@}{\Tdots@ORI}
  \newcommand{\Mdots@}{\@xp\mdots@}
  \AtBeginDocument{\DeclareRobustCommand{\dots}{\relax
                   \csname\ifmmode M\else T\fi dots@\endcsname}%
                   \ifx\@xp\@undefined\let\@xp\relax\fi
                   \ifx\mdots@\@undefined\let\Tdots@ORI\textellipsis
                                         \let\Mdots@\mathellipsis\fi}
  \def\bbl@frenchdots{\let\Tdots@\Frtextellipsis}
  \def\bbl@nonfrenchdots{\let\Tdots@\Tdots@ORI}
  \expandafter\addto\csname extras\CurrentOption\endcsname{%
    \bbl@frenchdots}
  \expandafter\addto\csname noextras\CurrentOption\endcsname{%
    \bbl@nonfrenchdots}
\fi
```

## 29.10 Global layout

In multilingual documents, some typographic rules must depend on the current language (e.g., hyphenation, typesetting of numbers, spacing before double punctuation…), others should, IMHO, be kept global to the document: especially the layout of lists (see 29.5) and the indentation of the first paragraph of sections (see 29.6).

`\FrenchLayout` `\StandardLayout` In a multilingual document, a unified look, either the French one or the standard one, can be assigned to the whole document: adding `\FrenchLayout` will give a global "French look" to the document (e.g., lists, regardless the current language, will be typeset as in French) and `\StandardLayout` forces a global "US-English" look. Both of these commands can only be used *before* the `\begin{document}`. `\StandardLayout` can also be of interest for class designers who do not want `frenchb` to interfere with their global layout choices.

As of version 1.4, `\FrenchLayout` and `\StandardLayout` only affect:

- the layout of lists (see 29.5),
- the indentation of the first paragraph of sections (see 29.6).

```
\ifLaTeXe
  \newcommand{\FrenchLayout}{%
    \renewcommand{\StandardLayout}{%
      \PackageWarning{frenchb.ldf}%
        {\protect\StandardLayout\space makes no change\MessageBreak
         (\protect\FrenchLayout\space has been selected before
         \MessageBreak for the whole document)}%
    }%
    \AtBeginDocument{\bbl@frenchitemize\bbl@frenchlistspacing
                     \bbl@frenchindent}%
    \let\bbl@nonfrenchitemize\relax
    \let\bbl@nonfrenchlistspacing\relax
    \let\bbl@nonfrenchindent\relax}
  \newcommand{\StandardLayout}{%
    \renewcommand{\FrenchLayout}{%
      \PackageWarning{frenchb.ldf}%
        {\protect\FrenchLayout\space makes no change\MessageBreak
         (\protect\StandardLayout\space has been selected before
         \MessageBreak for the whole document)}%
    }%
    \let\@ltiORI\labelitemi
    \let\@ltiiORI\labelitemii
    \let\@ltiiiORI\labelitemiii
    \let\@ltivORI\labelitemiv
    \let\itemizeORI\itemize
    \let\bbl@frenchitemize\relax
    \let\bbl@frenchlistspacing\relax
    \let\bbl@frenchindent\relax}
  \@onlypreamble\FrenchLayout
  \@onlypreamble\StandardLayout
\fi
```

## 29.11 Extra utilities

All that is left to do now is to provide the French user with some extra utilities.

`\up` `\ieme` `\up` eases the typesetting of superscripts like '1$^{\text{er}}$'. `\up` relies on `\textsuperscript` when available (i. e., in LaTeX 2$_\varepsilon$).

`\up@size` The internal macro `\up@size` holds the size at which the superscript will be typeset. The reason for this is that we have to specify it differently for different formats.

```
\ifx\sevenrm\@undefined
  \ifx\@ptsize\@undefined
    \let\up@size\small
  \else
    \ifx\selectfont\@undefined
```

In this case the format is the original LaTeX-2.09:

```
      \ifcase\@ptsize
        \let\up@size\ixpt\or
        \let\up@size\xpt\or
        \let\up@size\xipt
      \fi
```

When `\selectfont` is defined we probably have NFSS available:

```
    \else
      \ifcase\@ptsize
        \def\up@size{\fontsize\@ixpt{10pt}\selectfont}\or
        \def\up@size{\fontsize\@xpt{11pt}\selectfont}\or
        \def\up@size{\fontsize\@xipt{12pt}\selectfont}
      \fi
    \fi
  \fi
\else
```

If we end up here it must be a plain based TeX format, so:

```
    \let\up@size\sevenrm
\fi
```

Now we can define `\up`. When LaTeX $2_\varepsilon$ runs in compatibility mode (LaTeX-2.09 emulation), `\textsuperscript` is also defined, but does no good job, so we give two different definitions for `\up` using `\if@Two@E`.

```
\if@Two@E
  \DeclareRobustCommand*{\up}[1]{\textsuperscript{#1}}
\else
  \DeclareRobustCommand*{\up}[1]{\leavevmode\raise1ex\hbox{\up@size#1}}
\fi
```

`\ieme` is provided for compatibility with `francais.sty`, the other 5 for compatibility with `french.sty`:

```
\def\ieme{\up{\lowercase{e}}\xspace}
\def\iemes{\up{\lowercase{es}}\xspace}
\def\ier{\up{\lowercase{er}}\xspace}
\def\iers{\up{\lowercase{ers}}\xspace}
\def\iere{\up{\lowercase{re}}\xspace}
\def\ieres{\up{\lowercase{res}}\xspace}
```

`\No` `\no` `\primo` `\fprimo` And some more macros for numbering, first two support macros.

```
\DeclareRobustCommand*{\FrenchEnumerate}[1]{%
                          #1\up{\lowercase{o}}\kern+.3em}
\DeclareRobustCommand*{\FrenchPopularEnumerate}[1]{%
                          #1\up{\lowercase{o}})\kern+.3em}
```

Typing `\primo` should result in ‘1$^{o}$ ’,

```
\def\primo{\FrenchEnumerate1}
\def\secundo{\FrenchEnumerate2}
\def\tertio{\FrenchEnumerate3}
\def\quarto{\FrenchEnumerate4}
```

while typing `\fprimo)` gives ‘1$^{o}$) .

```
\def\fprimo){\FrenchPopularEnumerate1}
\def\fsecundo){\FrenchPopularEnumerate2}
\def\ftertio){\FrenchPopularEnumerate3}
\def\fquarto){\FrenchPopularEnumerate4}
```

Let’s provide two macros for the common abbreviations of “Numéro”.

```
\DeclareRobustCommand*{\No}{N\up{\lowercase{o}}\kern+.2em}
\DeclareRobustCommand*{\no}{n\up{\lowercase{o}}\kern+.2em}
```

`\bsc` As family names should be written in small capitals and never be hyphenated, we provide a command (its name comes from Boxed Small Caps) to input them easily; this is a simpler implementation of commands `\fsc` and `\lsc` from `french.sty` : no automatic uppercase/lowercase conversion is performed. Usage: `Jean~\bsc{Duchemin}`.

```
\DeclareRobustCommand*{\bsc}[1]{\leavevmode\hbox{\scshape #1}}
```

Some definitions for special characters. The first eight are mandatory for `\oe` etc. to work properly in moving arguments, the others just for convenience. We won’t define `\tilde` as a Text Symbol not to conflict with the macro `\tilde` for math mode and use the name `\tild` instead. Note that `\boi` may *not* be used in math mode, its name in math mode is `\backslash`.

`\degre` needs a special treatment: it is `\char6` in T1-encoding and `\char23` in OT1-encoding, both can be accessed by the command `\r{}` for ring accent.

```
\ifLaTeXe
  \DeclareTextSymbol{\ae}{T1}{230}
  \DeclareTextSymbol{\ae}{OT1}{26}
  \DeclareTextSymbol{\oe}{T1}{247}
  \DeclareTextSymbol{\oe}{OT1}{27}
  \DeclareTextSymbol{\AE}{T1}{198}
  \DeclareTextSymbol{\AE}{OT1}{29}
  \DeclareTextSymbol{\OE}{T1}{215}
  \DeclareTextSymbol{\OE}{OT1}{30}
  \DeclareTextSymbol{\at}{T1}{64}
  \DeclareTextSymbol{\at}{OT1}{64}
  \DeclareTextSymbol{\circonflexe}{T1}{94}
  \DeclareTextSymbol{\circonflexe}{OT1}{94}
  \DeclareTextSymbol{\tild}{T1}{126}
  \DeclareTextSymbol{\tild}{OT1}{126}
  \DeclareTextSymbolDefault{\at}{OT1}
  \DeclareTextSymbolDefault{\circonflexe}{OT1}
  \DeclareTextSymbolDefault{\tild}{OT1}
  \DeclareRobustCommand*{\boi}{\textbackslash}
  \DeclareRobustCommand*{\degre}{\r{}}
\else
  \def\T@one{T1}
  \ifx\f@encoding\T@one
    \newcommand{\degre}{\char6}
  \else
    \newcommand{\degre}{\char23}
  \fi
  \newcommand{\at}{\char64}
  \newcommand{\circonflexe}{\char94}
  \newcommand{\tild}{\char126}
  \newcommand{\boi}{{$\backslash$}}
\fi
```

`\degres` We now define a macro `\degres` for typesetting the abbreviation for 'degrees' (as in 'degrees Celsius'). As the bounding box of the character 'degree' has *very* different widths in CM/EC and PostScript fonts, we fix the width of the bounding box of `\degres` to 0.3 em, this lets the symbol 'degree' stick to the preceding (e.g., `45\degres`) or following character (e.g., `20~\degres C`).

If the TEX Companion fonts are available (`textcomp.sty`), we pick up `\textdegree` from them instead of using emulating 'degrees' from the `\r{}` accent. Otherwise we overwrite the (poor) definition of `\textdegree` given in `latin1.def`, `applemac.def` etc. (called by `inputenc.sty`) by our definition of `\degres`. We also advice the user (once only) to use TS1-encoding.

```
\ifLaTeXe
  \def\Warning@degree@TSone{%
        \PackageWarning{frenchb.ldf}{%
           Degrees would look better in TS1-encoding:
           \MessageBreak add \protect
           \usepackage{textcomp} to the preamble.
           \MessageBreak Degrees used}}
  \AtBeginDocument{\expandafter\ifx\csname M@TS1\endcsname\relax
                     \DeclareRobustCommand*{\degres}{%
                       \leavevmode\hbox to 0.3em{\hss\degre\hss}%
                       \Warning@degree@TSone
                       \global\let\Warning@degree@TSone\relax}%
                      \let\textdegree\degres
                   \else
                     \DeclareRobustCommand*{\degres}{%
                         \hbox{\UseTextSymbol{TS1}{\textdegree}}}%
                   \fi}
\else
  \DeclareRobustCommand*{\degres}{%
    \leavevmode\hbox to 0.3em{\hss\degre\hss}}
\fi
```

The following macros are used in the redefinition of `\^` and `\"` to handle the letter i: they allow users to type simply `\^i` and `\"i` instead of `\^{\i}` and `\"{\i}`.

MlTeX's macros dealing with accents conflict with those of LaTeX 2$_\varepsilon$, so we check whether `\csubinverse` is defined or not. If `\csubinverse` is *defined*, we are in MlTeX.

```
\ifLaTeXe
  \AtBeginDocument{%
    \ifx\csubinverse\@undefined
      \DeclareTextCompositeCommand{\^}{OT1}{i}{\^\i}%
      \DeclareTextCompositeCommand{\"}{OT1}{i}{\"\i}%
    \fi}
\fi
```

## 29.12 Date and clean up

`\datefrench` The macro `\datefrench` redefines the command `\today` to produce French dates.

```
\@namedef{date\CurrentOption}{%
  \def\today{\number\day
    \ifnum1=\day {\ier}\fi
    \space \ifcase\month
    \or janvier\or f\'evrier\or mars\or avril\or mai\or juin\or
    juillet\or ao\^ut\or septembre\or octobre\or novembre\or
    d\'ecembre\fi
    \space \number\year}}
```

The macro `\ldf@quit` takes care for setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value. The config file searched for has to be 'frenchb.cfg', and `\CurrentOption` has been set to 'french', so `\ldf@finish\CurrentOption` cannot be used: we first load 'frenchb.cfg' (in LaTeX2e only), then call `\ldf@quit\CurrentOption`. The macro-space used by some control sequences we do not need any longer, is freed.

```
\ifLaTeXe
  \loadlocalcfg{frenchb}
\fi
\let\T@one\relax
\let\@FI@\relax
\let\ifLaTeX\@undefined
\let\LaTeXtrue\@undefined
\let\LaTeXfalse\@undefined
\let\ifLaTeXe\@undefined
\let\LaTeXetrue\@undefined
\let\LaTeXefalse\@undefined
\ldf@quit\CurrentOption
⟨/code⟩
```

## 30 The Italian language

The file `italian.dtx`[26] defines all the language-specific macros for the Italian language.

The features of this language definition file are the following:

1. The Italian hyphenation is invoked, provided that file `ithyph.tex` was loaded when the LaTeX 2$_\varepsilon$ format was built; in case it was not, read the information coming with your distribution of the TeX software, and the babel documentation.

2. The language dependent fixed words to be inserted by such commands as `\chapter`, `\caption`, `\tableofcontents`, etc. are redefined in accordance with the Italian typographical practice.

3. Since Italian can be easily hyphenated and Italian practice allows to break a word before the last two letters, hyphenation parameters have been set accordingly, but a very high demerit value has been set in order to avoid word breaks in the penultimate line of a paragraph. Specifically the `\clubpenalty`, and the `\widowpenalty` are set to rather high values and `\finalhyphendemerits` is set to such a high value that hyphenation is prohibited between the last two lines of a paragraph.

4. Some language specific shortcuts have been defined so as to allow etymological hyphenation, specifically `"` inserts a break point in any word boundary that the typesetter chooses, provided it is not followed by and accented letter (very unlikely in Italian, where compulsory accents fall only on the last and ending vowel of a word, but may take place with compound words that include foreign roots), and `"|` when the desired break point falls before an accented letter.

5. The shortcut `""` introduces the raised (English) opening double quotes; this shortcut proves its usefulness when one reminds that the Italian keyboard misses the backtick key, and the backtick on a Windows based platform may be obtained only by pressing the `Alt` key while inputting the numerical code 0096; very, very annoying!

6. The shortcuts `"<` and `">` insert the French guillemots, sometimes used in Italian typography; with the T1 font encoding the ligatures `<<` and `>>` should insert such signs directly, but not all the virtual fonts that claim to follow the T1 font encoding actually contain the guillemots; with the OT1 encoding the guillemots are not available and must be faked in some way. By using the `"<` and `">` shortcuts (even with the T1 encoding) the necessary tests are performed and in case the suitable glyphs are taken from other fonts normally available with any good, modern LaTeX distribution.

[26]The file described in this section has version number v1.2q and was last revised on 2005/02/05. The original author is Maurizio Codogno, (`mau@beatles.cselt.stet.it`). It has been largely revised by Johannes Braams and Claudio Beccari

7. Three new specific commands `\unit`, `\ped`, and `\ap` are introduced so as to enable the correct composition of technical mathematics according to the ISO 31/XI recommendations. `\unit` does not get redefined if the babel package is loaded *after* the package `units.sty` whose homonymous command plays a different role and follows a different syntax.

For this language a limited number of shortcuts has been defined, table 6, some of which are used to overcome certain limitations of the Italian keyboard; in section 30.3 there are other comments and hints in order to overcome some other keyboard limitations.

| | |
|---|---|
| `"` | inserts a compound word mark where hyphenation is legal; it allows etymological hyphenation which is recommended for technical terms, chemical names and the like; it does not work if the next character is represented with a control sequence or is an accented character. |
| `"\|` | the same as the above without the limitation on characters represented with control sequences or accented ones. |
| `""` | inserts open quotes “. |
| `"<` | inserts open guillemots. |
| `">` | inserts closed guillemots. |
| `"/` | equivalent to `\slash` |

Table 6: Shortcuts for the Italian language

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
30.1 ⟨*code⟩
30.2 \LdfInit{italian}{captionsitalian}%
```

When this file is read as an option, i.e. by the `\usepackage` command, `italian` will be an ‘unknown’ language in which case we have to make it known. So we check for the existence of `\l@italian` to see whether we have to do something here.

```
30.3 \ifx\l@italian\@undefined
30.4     \@nopatterns{Italian}%
30.5     \adddialect\l@italian0\fi
```

The next step consists of defining commands to switch to (and from) the Italian language.

`\captionsitalian` The macro `\captionsitalian` defines all strings used in the four standard document classes provided with LaTeX.

```
30.6 \addto\captionsitalian{%
30.7   \def\prefacename{Prefazione}%
30.8   \def\refname{Riferimenti bibliografici}%
\def\abstractname{Sommario}%
\def\bibname{Bibliografia}%
\def\chaptername{Capitolo}%
\def\appendixname{Appendice}%
\def\contentsname{Indice}%
\def\listfigurename{Elenco delle figure}%
\def\listtablename{Elenco delle tabelle}%
\def\indexname{Indice analitico}%
\def\figurename{Figura}%
\def\tablename{Tabella}%
\def\partname{Parte}%
\def\enclname{Allegati}%
\def\ccname{e~p.~c.}%
\def\headtoname{Per}%
\def\pagename{Pag.}%    % in Italian the abbreviation is preferred
\def\seename{vedi}%
\def\alsoname{vedi anche}%
\def\proofname{Dimostrazione}%
\def\glossaryname{Glossario}%
}%
```

`\dateitalian` The macro `\dateitalian` redefines the command `\today` to produce Italian dates.

```
\def\dateitalian{%
\def\today{\number\day~\ifcase\month\or
  gennaio\or febbraio\or marzo\or aprile\or maggio\or giugno\or
  luglio\or agosto\or settembre\or ottobre\or novembre\or
  dicembre\fi\space \number\year}}%
```

`\italianhyphenmins` The italian hyphenation patterns can be used with both `\lefthyphenmin` and `\righthyphenmin` set to 2.

```
\providehyphenmins{\CurrentOption}{\tw@\tw@}
```

`\extrasitalian` `\noextrasitalian` Lower the chance that clubs or widows occur.

```
\addto\extrasitalian{%
\babel@savevariable\clubpenalty
\babel@savevariable\widowpenalty
\clubpenalty3000\widowpenalty3000}%
```

Never ever break a word between the last two lines of a paragraph in italian texts.

```
\addto\extrasitalian{%
\babel@savevariable\finalhyphendemerits
\finalhyphendemerits50000000}%
```

In order to enable the hyphenation of words such as “nell’altezza” we give the `'` a non-zero lower case code. When we do that TEX finds the following hyphenation points `nel-l'al-tez-za` instead of none.

```
\addto\extrasitalian{%
\lccode`'=`'}%
```

```
\addto\noextrasitalian{%
  \lccode`'=0}%
```

## 30.1 Support for etymological hyphenation

In his article on Italian hyphenation [1] Beccari pointed out that the Italian language gets hyphenated on a phonetic basis, although etymological hyphenation is allowed; this is in contrast with what happens in Latin, for example, where etymological hyphenation is always used. Since the patterns for both languages would become too complicated in order to cope with etymological hyphenation, in his paper Beccari proposed the definition of an active character '`_`' such that it could insert a "soft" discretionary hyphen at the compound word boundary. For several reasons that idea and the specific active character proved to be unpractical and was abandoned.

This problem is so important with the majority of the European languages, that babel from the very beginning developed the tradition of making the `"` character active so as to perform several actions that turned useful with every language. One of these actions consisted in defining the shortcut `"|` that was extensively used in German and in many other languages in order to insert a discretionary hyphen such that hyphenation would not be precluded in the rest of the word as it happens with the standard TeX command `\-`.

Meanwhile the `ec` fonts with the double Cork encoding (thus formerly called the `dc` fonts) have become more or less standard and are widely used by virtually all Europeans that write languages with many special national characters; by so doing they avoid the use of the `\accent` primitive which would be required with the standard `cm` fonts; with the latter fonts the primitive command `\accent` is such that hyphenation becomes almost impossible, in any case strongly impeached.

The `ec` fonts contain a special character, named "compound word mark", that occupies position 23 in the font scheme and may be input with the sequence `^^W`. Up to now, apparently, this special character has never been used in a practical way for the typesetting of languages rich of compound words; also it has never been inserted in the hyphenation pattern files of any language. Beccari modified his pattern file `ithyph.tex v4.8b` for Italian so as to contain five new patterns that involve `^^W`, and he tried to give the babel active character `"` a new shortcut definition, so as to allow the insertion of the "compound word mark" in the proper place within any word where two semantic fragments join up. With such facility for marking the compound word boundaries, etymological hyphenation becomes possible even if the patterns know nothing about etymology (but the typesetter hopefully does!). In Italian such etymological hyphenation is desirable with technical terms, chemical names, and the like.

Even this solution proved to be inconvenient on certain UN*X platforms, so Beccari resorted to another approach that uses the babel active character `"` and relies on the category code of the character that follows `"`.

```
\initiate@active@char{"}%
\addto\extrasitalian{\bbl@activate{"}\languageshorthands{italian}}%
```

\it@cwm The active character " is now defined for language `italian` so as to perform different actions in math mode compared to text mode; specifically in math mode a double quote is inserted so as to produce a double prime sign, while in text mode the temporary macro `\it@next` is defined so as to defer any further action until the next token category code has been tested.

```
\declare@shorthand{italian}{"}{%
\ifmmode
    \def\it@next{''}%
\else
    \def\it@next{\futurelet\it@temp\it@cwm}%
\fi
\it@next
}%
```

\it@cwm The `\it@next` service control sequence is such that upon its execution a temporary variable `\it@temp` is made equivalent to the next token in the input list without actually removing it. Such temporary token is then tested by the macro `\it@cwm` and if it is found to be a letter token, then it introduces a compound word separator control sequence `\it@allowhyphens` whose expansion introduces a discretionary hyphen and an unbreakable space; in case the token is not a letter, then it is tested against $|_{12}$: if so a compound word separator is inserted and the | token is removed, otherwise another test is performed so as to see if another double quote sign follows: in this case a double open quote mark is inserted, otherwise two other tests are performed so as to see if guillemets have to be inserted, otherwise nothing is done. The double quote shortcut for inserting a double open quote sign is useful for people who are inputting Italian text by means of an Italian keyboard that unfortunately misses the grave or backtick key. By this shortcut "" becomes equivalent to `` for inserting raised open high double quotes.

```
\def\it@@cwm{\nobreak\discretionary{-}{}{}\hskip\z@skip}%
\def\it@@ocap#1{\it@ocap}\def\it@@ccap#1{\it@ccap}%
\DeclareRobustCommand*{\it@cwm}{\let\it@@next\relax
\ifcat\noexpand\it@temp a%
    \def\it@@next{\it@@cwm}%
\else
    \if\noexpand\it@temp \string|%
        \def\it@@next{\it@@cwm\@gobble}%
    \else
        \if\noexpand\it@temp \string<%
            \def\it@@next{\it@@ocap}%
        \else
            \if\noexpand\it@temp \string>%
                \def\it@@next{\it@@ccap}%
            \else
                \if\noexpand\it@temp\string/%
                    \def\it@next{\slash\@gobble}%
                \else
                    \ifx\it@temp"%
                        \def\it@@next{``\@gobble}%
                        \fi
                    \fi
                \fi
            \fi
        \fi
    \fi
\it@@next}%
```

By this definition of `"` if one types `macro"istruzione` the possible break points become ma-cro-istru-zio-ne, while without the `"` mark they would be ma-croi-stru-zio-ne, according to the phonetic rules such that the `macro` prefix is not taken as a unit. A chemical name such as `des"clor"fenir"amina"cloridrato` is breakable as des-clor-fe-nir-ami-na-clo-ri-dra-to instead of de-sclor-fe-ni-ra-mi-na-...

In other language description files a shortcut is defined so as to allow a break point without actually inserting any hyphen sign; examples are given such as entrada/salida; actually if one wants to allow a breakpoint after the slash, it is much clearer to type `\slash` instead of `/` and LaTeX does everything by itself; here the shortcut `"/` was introduced to stand for `\slash` so that one can type `input"/output` and allow a line break after the slash. This shortcut works only for the slash, since in Italian such constructs are extremely rare.

Attention: the expansion of `"` takes place before the actual expansion of OT1 or T1 accented sequences such as `` \`{a} ``; therefore this etymological hyphenation facility works as it should only when the semantic word fragments *do not start* with an accented letter; this in Italian is always avoidable, because compulsory accents fall only on the last vowel, but it may be necessary to mark a compound word where one or more components come from a foreign language and contain diacritical marks according to the spelling rules of that language. In this case the special shorthand `"|` may be used that performs exactly as `"` normally does, except that the `|` sign is removed from the token input list: `kilo"|{\"o}rsted` gets hyphenated as `ki-lo-ör-sted`.

## 30.2 Facilities required by the ISO 31/XI regulations

The ISO 31/XI regulations require that units of measure are typeset in upright font in any circumstance, math or text, and that in text mode they are separated from the numerical indication of the measure with an unbreakable (thin) space. The command `\unit` that was defined for achieving this goal happened to conflict with the homonymous command defined by the package `units.sty`; we therefore need to test if that package has already been loaded so as to avoid conflicts; we assume that if the user loads that package, s/he wants to use that package facilities and command syntax.

The same regulations require also that super and subscripts (apices and pedices) are in upright font, *not in math italics*, when they represent “adjectives” or appositions to mathematical or physical variables that do not represent countable or measurable entities such as, for example, $V_{\mathrm{max}}$ or $V_{\mathrm{rms}}$ for a maximum or a root mean square voltage, compared to $V_i$ or $V_T$ as the $i$-th voltage in a set, or a voltage that depends on the thermodynamic temperature $T$. See [2] for a complete description of the ISO regulations in connection with typesetting.

More rarely it happens to use superscripts that are not mathematical variables, such as the notation $\mathbf{A}^\mathrm{T}$ to denote the transpose of matrix $\mathbf{A}$; text superscripts are useful also as ordinals or in old fashioned abbreviations in text mode; for example the feminine ordinal $1^\mathrm{a}$ or the old fashioned obsolete abbreviation $\mathrm{F}^\mathrm{lli}$ for Fratelli in company names (compare with "Bros." for $\underline{\text{bro}}$ther$\underline{\text{s}}$ in American English); text subscripts are mostly used in logos.

`\unit` `\ap` `\ped` First we define the new (internal) commands `\bbl@unit`, `\bbl@ap`, and `\bbl@ped` as robust ones.

```
\@ifpackageloaded{units}{}{%
  \DeclareRobustCommand*{\bbl@unit}[1]{%
    \textormath{\,\mbox{#1}}{\,\mathrm{#1}}}%
  }%
\DeclareRobustCommand*{\bbl@ap}[1]{%
  \textormath{\textsuperscript{#1}}{^{\mathrm{#1}}}}%
\DeclareRobustCommand*{\bbl@ped}[1]{%
  \textormath{$_{\mbox{\fontsize\sf@size\z@
        \selectfont#1}}$}{_\mathrm{#1}}}%
```

Then we can use `\let` to define the user level commands, but in case the macros already have a different meaning before entering in Italian mode typesetting, we first memorize their meaning so as to restore them on exit.

```
\@ifpackageloaded{units}{}{%
  \addto\extrasitalian{%
    \babel@save\unit\let\unit\bbl@unit}%
  }%
\addto\extrasitalian{%
  \babel@save\ap\let\ap\bbl@ap
  \babel@save\ped\let\ped\bbl@ped
  }%
```

## 30.3 Accents

Most of the other language description files introduce a number of shortcuts for inserting accents and other language specific diacritical marks in a more comfortable way compared with the lengthy standard TEX conventions. When an Italian keyboard is being used on a Windows based platform, it exhibits such limitations that up to now no convenient shortcuts have been developed; the reason lies in the fact that the Italian keyboard lacks the grave accent (also known as "backtick"), which is compulsory on all accented vowels except the 'e', but, on the opposite, it carries the keys with all the accented lowercase vowels; the keyboard lacks also the tie ~ (tilde) key, while the curly braces require pressing three keys simultaneously.

The best solution Italians have found so far is to use a smart editor that accepts shortcut definitions such that, for example, by striking `"(` one gets directly `{` on the screen and the same sign is saved into the `.tex` file; the same smart editor should be capable of translating the accented characters into the standard TEX sequences when writing a file to disk (for the sake of file portability), and to transform the standard TEX sequences into the corresponding signs when loading a `.tex` file from disk to memory. Such smart editors do exist and can be downloaded from the CTAN archives.

For what concerns the missing backtick key, which is used also for inputting the open quotes, it must be noticed that the shortcut `""` described above completely solves the problem for *double* raised open quotes; according to the traditions of particular publishing houses, since there are no compulsory regulations on the matter, the French guillemets may be used; in this case the T1 font encoding solves the problem by means of its built in ligatures `<<` and `>>`. But...

## 30.4 *Caporali* or French double quotes

Although the T1 font encoding ligatures solve the problem, there are some circumstances where even the T1 font encoding cannot be used, either because the author/typesetter wants to use the OT1 encoding, or because s/he uses a font set that does not comply completely with the T1 font encoding; some virtual fonts, for example, are supposed to implement the double Cork font encoding but actually miss some glyphs; one such virtual font set is given by the `ae` virtual fonts, because they are supposed to implement such double font encoding simply using the `cm` fonts, of which the type 1 PostScript version exists and is freely available. Since guillemets (in Italian *caporali*) do not exist in any `cm` latin font, their glyphs must be substituted with something else that approaches them.

Since in French typesetting guillemets are compulsory, the French language definition file resorts to a clever font substitution; such file exploits the LATEX $2_\varepsilon$ font selection machinery so as to get the guillemets from the Cyrillic fonts, because it suffices to locally change the default encoding. There are several sets of Cyrillic fonts, but the ones that obey the OT2 font encoding are generally distributed with all recent implementations of the TEX software; they are part of the American Mathematical Society fonts and come both as METAFONT source files and Type 1 PostScript `.pfb` files. The availability of such fonts should be guaranteed by the presence of the `OT2cmr.fd` font description file. Actually the presence of this file does not guarantee the completeness of your TEX implementation; should LATEX complain about a missing Cyrillic `.tfm` file (that kind of file that contains the font metric parameters) and/or about missing Cyrillic (.mf) files, then your TEX system is *incomplete* and you should download such fonts from the CTAN archives. Temporarily you may issue the command `\LtxSymbCaporali` so as to approximate the missing glyphs with the LATEX symbol fonts. In some case warning messages are issued so as to inform the typesetter about the necessity of resorting to some *poor man* solution.

In spite of these alternate fonts, we must avoid invoking unusual fonts if the available encoding allows to use built in caporali. As far as I know (CB) the only T1-encoded font families that miss the guillemets are the AE ones; we therefore first test if the default encoding id the T1 one and in this case if the AE families are the default ones; in order for this to work properly it is necessary to load these optional packages *before* `babel`. If the T1 encoding is not the default one when the

Italian language is specified, then some substitutions must be done.

\LtxSymbCaporali \it@ocap \it@ccap We define some macros for substituting the default guillemets; first the emulation by means of the LaTeX symbols; each one of these macro sets actually redefines the control sequences `\it@ocap` and `\it@ccap` that are the ones effectively activated by the shortcuts `"<` and `">`.

```
\def\LtxSymbCaporali{%
    \DeclareRobustCommand*{\it@ocap}{\mbox{%
       \fontencoding{U}\fontfamily{lasy}\selectfont(\kern-0.20em(}%
       \ignorespaces}%
    \DeclareRobustCommand*{\it@ccap}{\ifdim\lastskip>\z@\unskip\fi
    \mbox{%
       \fontencoding{U}\fontfamily{lasy}\selectfont)\kern-0.20em)}}%
}%
```

Then the substitution with any specific font that contains such glyphs; it might be the CBgreek fonts, the Cyrillic one, the super-cm ones, the lm ones, or any other the user might prefer (the code is adapted from the one that appears in the `frenchb.ld` file; thanks to Daniel Flipo). By default if the user did not select the T1 encoding, the existence of the CBgreek fonts is tested; if they exist the guillemets are taken from this font, and since its families are a superset of the default CM ones and they apply also to typeset slides with the standard class `slides`. If the CBgreek fonts are not found, then the existence of the Cyrillic ones is tested, although this choice is not suited for typesetting slides; otherwise the poor man solution of the LaTeX special symbols is used. In any case the user can force the use of the Cyrillic guillemets substitution by issuing the declaration `\CyrillicCaporali` just before the `\begin{document}` statement; in alternative the user can specify with

`\CaporaliFrom{`⟨*encoding*⟩`}{`⟨*family*⟩`}{`⟨*opening number*⟩`}{`⟨*closing number*⟩`}`

the encoding and family of the font s/he prefers, and the slot numbers of the opening and closing guillemets respectively. For example if the T1-encoded Latin Modern fonts are desired the specific command should be

`\CaporaliFrom{T1}{lmr}{19}{20}`

These user choices might be necessary for assuring the correct typesetting with fonts that contain the required glyphs and are available also in PostScript form so as to use them directly di `pdflatex`, for example.

```
\def\CaporaliFrom#1#2#3#4{%
  \DeclareFontEncoding{#1}{}{}%
  \DeclareTextCommand{\it@ocap}{T1}{%
    {\fontencoding{#1}\fontfamily{#2}\selectfont\char#3\ignorespaces}}%
  \DeclareTextCommand{\it@ccap}{T1}{\ifdim\lastskip>\z@\unskip\fi%
    {\fontencoding{#1}\fontfamily{#2}\selectfont\char#4}}%
  \DeclareTextCommand{\it@ocap}{OT1}{%
    {\fontencoding{#1}\fontfamily{#2}\selectfont\char#3\ignorespaces}}%
  \DeclareTextCommand{\it@ccap}{OT1}{\ifdim\lastskip>\z@\unskip\fi%
    {\fontencoding{#1}\fontfamily{#2}\selectfont\char#4}}}
```

Then we set a boolean variable and test the default family; if such family has a name that starts with the letters "ae" then we have no built in guillemets; of course if the AE font family is chosen after the babel package is loaded, the test does not perform as required.

```
\def\get@ae#1#2#3!{\def\bbl@ae{#1#2}}%
\def\@ifT@one@noCap{\expandafter\get@ae\f@family!%
\def\bbl@temp{ae}\ifx\bbl@ae\bbl@temp\expandafter\@firstoftwo\else
    \expandafter\@secondoftwo\fi}%
```

We set another couple of boolean variables for testing the existence of the CBgreek or the Cyrillic fonts

```
\newif\if@CBgreekEncKnown
\IfFileExists{lgrcmr.fd}%
      {\@CBgreekEncKnowntrue}{\@CBgreekEncKnownfalse}
\newif\if@CyrEncKnown
\IfFileExists{ot2cmr.fd}%
     {\@CyrEncKnowntrue}{\@CyrEncKnownfalse}%
```

\CBgreekCaporali
\CyrillicCaporali
\T@unoCaporali

Next we define the macros \CBgreekCaporali, \T@unoCaporali, and \CyrillicCaporali; with the latter one we test the loaded class, and if it's slides nothing gets done. In any case each one of these declarations, if used, must be specified in the preamble.

```
\def\CBgreekCaporali{\@ifclassloaded{slides}{%
      \IfFileExists{lgrlcmss.fd}{\DeclareFontEncoding{LGR}{}{}%
            \DeclareRobustCommand*{\it@ccap}%
                  {\ifdim\lastskip>\z@\unskip\fi
                        {\fontencoding{LGR}\selectfont))}}%
            \DeclareRobustCommand*{\it@ocap}%
                  {{\fontencoding{LGR}\selectfont((}\ignorespaces}}%
            {\LtxSymbCaporali}}%
      {\DeclareFontEncoding{LGR}{}{}%
      \DeclareRobustCommand*{\it@ccap}%
            {\ifdim\lastskip>\z@\unskip
            \fi{\fontencoding{LGR}\selectfont))}}%
      \DeclareRobustCommand*{\it@ocap}%
            {{\fontencoding{LGR}\selectfont((}\ignorespaces}}%
      }%
\def\CyrillicCaporali{\@ifclassloaded{slides}{\relax}%
      {\DeclareFontEncoding{OT2}{}{}%
      \DeclareRobustCommand*{\it@ccap}%
            {\ifdim\lastskip>\z@\unskip\fi
            {\fontencoding{OT2}\selectfont\char62\relax}}%
      \DeclareRobustCommand*{\it@ocap}%
            {{\fontencoding{OT2}\selectfont\char60\relax}\ignorespaces}}}%
\@onlypreamble{\CBgreekCaporali}\@onlypreamble{\CyrillicCaporali}%
\def\T@unoCaporali{\DeclareRobustCommand*{\it@ocap}{<<\ignorespaces}%
     \DeclareRobustCommand*{\it@ccap}{\ifdim\lastskip>\z@\unskip\fi>>}}%
```

Now we can do some real setting; first we start testing the encoding; if the encoding is T1 we test if the font family is the AE one; if so, we further test for other possibilities

```
\ifx\cf@encoding\bbl@t@one
  \@ifT@one@noCap{%
     \if@CBgreekEncKnown
        \CBgreekCaporali
     \else
        \if@CyrEncKnown
           \CyrilicCaporali
        \else
           \LtxSymbCaporali
        \fi
     \fi}%
     {\T@unoCaporali}%
```

But if the default encoding is not the T1 one, then the substitutions must be performed.

```
\else
     \if@CBgreekEncKnown
        \CBgreekCaporali
     \else
        \if@CyrEncKnown
           \CyrilicCaporali
        \else
           \LtxSymbCaporali
        \fi
     \fi
\fi
```

## 30.5 Finishing commands

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{italian}%
⟨/code⟩
```

# References

[1] Beccari C., "Computer Aided Hyphenation for Italian and Modern Latin", TUGboat vol. 13, n. 1, pp. 23-33 (1992).

[2] Beccari C., "Typesetting mathematics for science and technology according to ISO 31/XI", TUGboat vol. 18, n. 1, pp. 39-48 (1997).

## 31 The Latin language

The file `latin.dtx`[27] defines all the language-specific macros for the Latin language both in modern and medieval spelling.

For this language two “styles” of typesetting are implemented: “regular” or modern-spelling Latin, and medieval Latin. The medieval Latin specific commands can be activated by means of the language attribute `medieval`; the medieval spelling differs from the modern one by the systematic use of the lower case ‘u’ also where in modern spelling the letter ‘v’ is used; when typesetting with capital letters, on the opposite, the letter ’V’ is used also in place of ’U’. Medieval spelling also includes the ligatures `\ae` (æ), `\oe` (œ), `\AE` (Æ), and `\OE` (Œ) that are not used in modern spelling, nor were used in the classical times.

Furthermore a third typesetting style `withprosodicmarks` is defined in order to use special shortcuts for inserting breves and macrons when typesetting grammars, dictionaries, teaching texts, and the like, where prosodic marks are important for the complete information on the words or the verses. The shortcuts, listed in table 7 and described in section 32, may interfere with other packages; therefore by default this third style is off and no interference is introduced. If this third style is used and interference is experienced, there are special commands for turning on and off the specific short hand commands of this style.

For what concerns babel and typesetting with LaTeX, the differences between the two styles of spelling reveal themselves in the strings used to name for example the “Preface” that becomes “Praefatio” or “Præfatio” respectively. Hyphenation rules are also different, but the hyphenation pattern file `lahyph.tex` takes care of both versions of the language. Needless to say that such patterns must be loaded in the LaTeX format by by running `initex` (or whatever the name of the initializer) on `latex.ltx`.

The name strings for chapters, figures, tables, etcetera, are suggested by prof. Raffaella Tabacco, a classicist of the University of Turin, Italy, to whom we address our warmest thanks. The names suggested by Krzysztof Konrad Żelechowski, when different, are used as the names for the medieval variety, since he made a word and spelling choice more suited for this variety.

For this language some shortcuts are defined according to table 7; all of them are supposed to work with both spelling styles, except where the opposite is explicitly stated.

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
31.1 ⟨*code⟩
31.2 \LdfInit{latin}{captionslatin}
```

When this file is read as an option, i.e. by the `\usepackage` command, `latin` will be an ‘unknown’ language in which case we have to make it known. So we check for the existence of `\l@latin` to see whether we have to do something here.

[27]The file described in this section has version number v2.0f and was last revised on 2005/03/30. The original author is Claudio Beccari with contributions by Krzysztof Konrad Żelechowski, (`kkz@alfa.mimuw.edu.pl`)

| | |
|---|---|
| `^i` | inserts the breve accent as ĭ; valid also for the other lowercase vowels, but it does not operate on the medieval ligatures æ and œ. |
| `=a` | inserts the macron accent as ā; valid also for the other lowercase vowels, but it does not operate on the medieval ligatures æ and œ. |
| `"` | inserts a compound word mark where hyphenation is legal; the next character must not be a medieval ligature æ or œ, nor an accented letter (foreign names). |
| `"\|` | same as above, but operates also when the next character is a medieval ligature or an accented letter. |

Table 7: Shortcuts defined for the Latin language. The characters `^` and `=` are active only when the language attribute `withprosodicmarks` has been declared, otherwise they are disabled; see section 32 for more details.

```
\ifx\l@latin\@undefined
    \@nopatterns{Latin}
    \adddialect\l@latin0\fi
```

Now we declare the `medieval` language attribute.

```
\bbl@declare@ttribute{latin}{medieval}{%
  \addto\captionslatin{\def\prefacename{Pr{\ae}fatio}}%
  \def\november{Nouembris}%
  \expandafter\addto\expandafter\extraslatin
  \expandafter{\extrasmedievallatin}%
  }
```

The third typesetting style `withprosodicmarks` is defined here

```
\bbl@declare@ttribute{latin}{withprosodicmarks}{%
  \expandafter\addto\expandafter\extraslatin
  \expandafter{\extraswithprosodicmarks}%
  }
```

It must be remembered that the `medieval` and the `withprosodicmarks` styles may be used together.

The next step consists of defining commands to switch to (and from) the Latin language[28].

`\captionslatin` The macro `\captionslatin` defines all strings used in the four standard document classes provided with LaTeX.

```
\@namedef{captionslatin}{%
  \def\prefacename{Praefatio}%
  \def\refname{Conspectus librorum}%
  \def\abstractname{Summarium}%
  \def\bibname{Conspectus librorum}%
  \def\chaptername{Caput}%
```

[28]Most of these names were kindly suggested by Raffaella Tabacco.

```
  \def\appendixname{Additamentum}%
  \def\contentsname{Index}%
  \def\listfigurename{Conspectus descriptionum}%
  \def\listtablename{Conspectus tabularum}%
  \def\indexname{Index rerum notabilium}%
  \def\figurename{Descriptio}%
  \def\tablename{Tabula}%
  \def\partname{Pars}%
  \def\enclname{Adduntur}%   Or " Additur" ? Or simply Add.?
  \def\ccname{Exemplar}%     Use the recipient's dative
  \def\headtoname{\ignorespaces}% Use the recipient's dative
  \def\pagename{Charta}%
  \def\seename{cfr.}%
  \def\alsoname{cfr.}% R.Tabacco never saw "cfr. atque" or similar forms
  \def\proofname{Demonstratio}%
  \def\glossaryname{Glossarium}%
  }
```

In the above definitions there are some points that might change in the future or that require a minimum of attention from the typesetter.

1. the `\enclname` is translated by a passive verb, that literally means "(they) are being added"; if just one enclosure is joined to the document, the plural passive is not suited any more; nevertheless a generic plural passive might be incorrect but suited for most circumstances. On the opposite "Additur", the corresponding singular passive, might be more correct with one enclosure and less suited in general: what about the abbreviation "Add." that works in both cases, but certainly is less elegant?

2. The `\headtoname` is empty and gobbles the possible following space; in practice the typesetter should use the dative of the recipient's name; since nowadays not all such names can be translated into Latin, they might result indeclinable. The clever use of an appellative by the typesetter such as "Domino" or "Dominae" might solve the problem, but the header might get too impressive. The typesetter must make a decision on his own.

3. The same holds true for the copy recipient's name in the "Cc" field of `\ccname`.

`\datelatin` The macro `\datelatin` redefines the command `\today` to produce Latin dates; the choice of faked small caps Latin numerals is arbitrary and may be changed in the future. For medieval latin the spelling of 'Novembris' should be *Nouembris.* This is taken care of by using a control sequence which can be redefined when the attribute 'medieval' is selected.

```
\def\datelatin{%
  \def\november{Novembris}%
  \def\today{%
    {\check@mathfonts\fontsize\sf@size\z@\math@fontsfalse\selectfont
      \uppercase\expandafter{\romannumeral\day}}\nobreakspace
    \ifcase\month\or
    Ianuarii\or Februarii\or Martii\or Aprilis\or Maii\or Iunii\or
    Iulii\or Augusti\or Septembris\or Octobris\or \november\or
    Decembris\fi
    \space{\uppercase\expandafter{\romannumeral\year}}}}
```

`\romandate` Thomas Martin Widmann (`viralbus@daimi.au.dk`) developed a macro originally named `\latindate` (but to be renamed `\romandate` so as not to conflict with the standard babel conventions) that should compute and translate the current date into a date *ab urbe condita* with days numbered according to the kalendae and idus; for the moment this is a placeholder for Thomas' macro, waiting for a self standing one that keeps local all the intermediate data, counters, etc. If he succeeds, here is the place to add his macro.

`\latinhyphenmins` The Latin hyphenation patterns can be used with both `\lefthyphenmin` and `\righthyphenmin` set to 2.

```
\providehyphenmins{\CurrentOption}{\tw@\tw@}
```

`\extraslatin` `\noextraslatin` For this language the `\clubpenalty`, `\widowpenalty` are set to rather high values and `\finalhyphendemerits` is set to such a high value that hyphenation is prohibited between the last two lines of a paragraph.

```
\addto\extraslatin{%
  \babel@savevariable\clubpenalty
  \babel@savevariable\widowpenalty
  \clubpenalty3000\widowpenalty3000}
```

Never ever break a word between the last two lines of a paragraph in latin texts.

```
\addto\extraslatin{%
  \babel@savevariable\finalhyphendemerits
  \finalhyphendemerits50000000}
```

With medieval Latin we need the suitable correspondence between upper case V and lower case u, since in that spelling there is only one sign, and the u shape is the (uncial) version of the capital V. Everything else is identical with Latin.

```
\addto\extrasmedievallatin{%
\babel@savevariable{\lccode`\V}%
\babel@savevariable{\uccode`\u}%
\lccode`\V=`\u \uccode`\u=`\V}
```

`\SetLatinLigatures` We need also the lccodes for æ and œ; since they occupy different positions in the OT1 TEX-fontencoding compared to the T1 one, we must save the lc- and the uccodes for both encodings, but we specify the new lc- and uccodes separately as it appears natural not to change encoding while typesetting the same language. The encoding is assumed to be set before starting to use the Latin language, so that if Latin is the default language, the font encoding must be chosen before requiring the `babel` package with the `latin` option, in any case before any `\selectlanguage` or `\foreignlanguage` command.

All this fuss is made in order to allow the use of the medieval ligatures æ and œ while typesetting with the medieval spelling; I have my doubts that the medieval spelling should be used at all in modern books, reports, and the like, as the uncial 'u' shape of the lower case 'v' and the above ligatures were used until the middle of the XVII century. Since my opinion (CB) may be wrong, I managed to set up the instruments and it is up to the typesetter to use them or not.

```
\addto\extrasmedievallatin{%
  \babel@savevariable{\lccode`\^^e6}% T1   \ae
  \babel@savevariable{\uccode`\^^e6}% T1   \ae
  \babel@savevariable{\lccode`\^^c6}% T1   \AE
  \babel@savevariable{\lccode`\^^f7}% T1   \oe
  \babel@savevariable{\uccode`\^^f7}% T1   \OE
  \babel@savevariable{\lccode`\^^d7}% T1   \OE
  \babel@savevariable{\lccode`\^^1a}% OT1  \ae
  \babel@savevariable{\uccode`\^^1a}% OT1  \ae
  \babel@savevariable{\lccode`\^^1d}% OT1  \AE
  \babel@savevariable{\lccode`\^^1b}% OT1  \oe
  \babel@savevariable{\uccode`\^^1b}% OT1  \OE
  \babel@savevariable{\lccode`\^^1e}% OT1  \OE
  \SetLatinLigatures}
\providecommand\SetLatinLigatures{%
  \def\@tempA{T1}\ifx\@tempA\f@encoding
    \catcode`\^^e6=11 \lccode`\^^e6=`\^^e6 \uccode`\^^e6=`\^^c6 % \ae
    \catcode`\^^c6=11 \lccode`\^^c6=`\^^e6 % \AE
    \catcode`\^^f7=11 \lccode`\^^f7=`\^^f7 \uccode`\^^f7=`\^^d7 % \oe
    \catcode`\^^d7=11 \lccode`\^^d7=`\^^f7 % \OE
  \else
    \catcode`\^^1a=11 \lccode`\^^1a=`\^^1a \uccode`\^^1a=`\^^1d % \ae
    \catcode`\^^1d=11 \lccode`\^^1d=`\^^1a % \AE (^^])
    \catcode`\^^1b=11 \lccode`\^^1b=`\^^1b \uccode`\^^1b=`\^^1e % \oe
    \catcode`\^^1e=11 \lccode`\^^1e=`\^^1b % \OE (^^^)
  \fi
  \let\@tempA\@undefined
  }
```

With the above definitions we are sure that `\MakeUppercase` works properly and `\MakeUppercase{C{\ae}sar}` correctly 'yields 'CÆSAR"; correspondingly `\MakeUppercase{Heluetia}` correctly yields "HELVETIA".

## 32 Latin shortcuts

For writing dictionaries or didactic texts we defined a third language attribute, or a third typesetting style, where a couple of other active characters are defined: `^` for marking a vowel with the breve sign, and `=` for marking a vowel with the macro sign. Please take notice that neither the OT1 font encoding, nor the T1 one for most vowels, contain directly the marked vowels, therefore hyphenation of words containing these "accents" may become problematic; for this reason the above active characters not only introduce the required accent, but also the necessary code for allowing hyphenation in the rest of the word.

It must be remarked that the active characters `^` and `=` may have other meanings in other contexts. For example the equals sign is used by the graphic extensions for specifying keyword options for handling the graphic elements to be included in the document. At the same time, as mentioned in the previous paragraph, diacritical marking in Latin is used only for typesetting certain kind of documents, such as grammars and dictionaries. It is reasonable that the breve and macron active characters may be switched on and off at will, and in particular that they are off by default if the attribute `withprosodicmarks` has not been set.

`\ProsodicMarksOn` `\ProsodicMarksOff` We begin by adding to the normal typesetting style the definitions of the new commands `\ProsodicMarksOn` and `\ProsodicMarksOff` that should produce error messages when the third style is not declared:

```
\addto\extraslatin{\def\ProsodicMarksOn{%
\GenericError{(latin)\@spaces\@spaces\@spaces\@spaces}%
            {Latin language error: \string\ProsodicMarksOn\space
            is defined by setting the\MessageBreak
            language attribute to ‘withprosodicmarks’\MessageBreak
            If you continue you are likely to encounter\MessageBreak
            fatal errors that I can’t recover}%
            {See the Latin language description in the babel
            documentation for explanation}{\@ehd}}}
\addto\extraslatin{\let\ProsodicMarksOff\relax}
```

Then we temporarily set the caret and the equals sign to active characters so that they can receive their definitions:

```
\catcode‘\= \active
\catcode‘\^ \active
```

and we add the necessary declarations to the macros that are being activated when the Latin language and its typesetting styles are declared:

```
\addto\extraslatin{\languageshorthands{latin}}%
\addto\extraswithprosodicmarks{\bbl@activate{^}}%
\addto\extraswithprosodicmarks{\bbl@activate{=}}%
\addto\noextraswithprosodicmarks{\bbl@deactivate{^}}%
\addto\noextraswithprosodicmarks{\bbl@deactivate{=}}%
\addto\extraswithprosodicmarks{\ProsodicMarks}
```

`\ProsodicMarks` Next we define the defining macro for the active characters

```
\def\ProsodicMarks{%
\def\ProsodicMarksOn{\catcode‘\^ 13\catcode‘\= 13\relax}%
\def\ProsodicMarksOff{\catcode‘\^ 7\catcode‘\= 12\relax}%
```

Notice that with the above redefinitions of the commands `\ProsodicMarksOn` and `\ProsodicMarksOff`, the operation of the newly defined shortcuts may be switched on and off at will, so that even if a picture has to be inserted in the document by means of the commands and keyword options of the `graphicx` package, it suffices to switch them off before invoking the picture including command, and switched on again afterwards; or, even better, since the picture very likely is being inserted within a `figure` environment, it suffices the switch them off within the environment, being conscious that their deactivation remains local to the environment.

```
\initiate@active@char{^}%
\initiate@active@char{=}%
\declare@shorthand{latin}{^a}{%
    \textormath{\u{a}\allowhyphens}{\hat{a}}}%
\declare@shorthand{latin}{^e}{%
    \textormath{\u{e}\allowhyphens}{\hat{e}}}%
\declare@shorthand{latin}{^i}{%
    \textormath{\u{\i}\allowhyphens}{\hat{\imath}}}%
\declare@shorthand{latin}{^o}{%
    \textormath{\u{o}\allowhyphens}{\hat{o}}}%
\declare@shorthand{latin}{^u}{%
    \textormath{\u{u}\allowhyphens}{\hat{u}}}%
\declare@shorthand{latin}{=a}{%
    \textormath{\={a}\allowhyphens}{\bar{a}}}%
\declare@shorthand{latin}{=e}{%
    \textormath{\={e}\allowhyphens}{\bar{e}}}%
\declare@shorthand{latin}{=i}{%
    \textormath{\={\i}\allowhyphens}{\bar{\imath}}}%
\declare@shorthand{latin}{=o}{%
    \textormath{\={o}\allowhyphens}{\bar{o}}}%
\declare@shorthand{latin}{=u}{%
    \textormath{\={u}\allowhyphens}{\bar{u}}}%
}
```

Notice that the short hand definitions are given only for lower case letters; it would not be difficult to extend the set of definitions to upper case letters, but it appears of very little use in view of the kind of documents where prosodic marks are supposed to be used. Nevertheless in those rare cases when it's required to set some uppercase letters with their prosodic marks, it is always possible to use the standard LaTeX commands such as `\u{I}` for typesetting Ĭ, or `\={A}` for typesetting Ā.

Finally we restore the caret and equals sign initial default category codes.

```
\catcode`\= 12\relax
\catcode`\^ 7\relax
```

It must be understood that by using the above prosodic marks, line breaking is somewhat impeached; since such prosodic marks are used almost exclusively in dictionaries, grammars, and poems (only in school textbooks), this shouldn't be of any importance for what concerns the quality of typesetting.

## 33 Etymological hyphenation

In order to deal in a clean way with prefixes and compound words to be divided etymologically, the active character `"` is given a special definition so as to behave as a discretionary break with hyphenation allowed after it. Most of the code for dealing with the active " is already contained in the core of babel, but we are going to use it as a single character shorthand for Latin.

```
\initiate@active@char{"}%
\addto\extraslatin{\bbl@activate{"}%
}
```

A temporary macro is defined so as to take different actions in math mode and text mode: specifically in the former case the macro inserts a double quote as it should in math mode, otherwise another delayed macro comes into action.

```
\declare@shorthand{latin}{"}{%
  \ifmmode
    \def\lt@@next{''}%
  \else
    \def\lt@@next{\futurelet\lt@temp\lt@cwm}%
  \fi
  \lt@@next
}%
```

In text mode the `\lt@next` control sequence is such that upon its execution a temporary variable `\lt@temp` is made equivalent to the next token in the input list without actually removing it. Such temporary token is then tested by the macro `\lt@cwm` and if it is found to be a letter token, then it introduces a compound word separator control sequence `\lt@allowhyphens` whose expansion introduces a discretionary hyphen and an unbreakable space; in case the token is not a letter, the token is tested again to find if it is the character |, in which case it is gobbled and a discretionary break is introduced.

```
\def\lt@allowhyphens{\nobreak\discretionary{-}{}{}\hskip\z@skip}
\newcommand*{\lt@cwm}{\let\lt@n@xt\relax
  \ifcat\noexpand\lt@temp a%
    \let\lt@n@xt\lt@allowhyphens
  \else
    \if\noexpand\lt@temp\string|%
      \def\lt@n@xt{\lt@allowhyphens\@gobble}%
    \fi
  \fi
  \lt@n@xt}%
```

Attention: the category code comparison does not work if the temporary control sequence `\lt@temp` has been let equal to an implicit character, such as `\ae`; therefore this etymological hyphenation facility does not work with medieval Latin spelling when " immediately precedes a ligature. in order to overcome this drawback the shorthand "| may be used in such cases; it behaves exactly as ", but it does not test the implicit character control sequence. An input such as `super"|{\ae}quitas`[29] gets hyphenated as `su-per-æqui-tas` instead of `su-pe-ræ-qui-tas`.

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

[29]This word does not exist in “regular” Latin, and it is used just as an example.

```
\ldf@finish{latin}
⟨/code⟩
```

## 34 The Portuguese language

The file `portuges.dtx`[30] defines all the language-specific macros for the Portuguese language as well as for the Brasilian version of this language.

For this language the character `"` is made active. In table 8 an overview is given of its purpose.

| | |
|---|---|
| `"\|` | disable ligature at this position. |
| `"-` | an explicit hyphen sign, allowing hyphenation in the rest of the word. |
| `""` | like `"-`, but producing no hyphen sign (for words that should break at some sign such as "entrada/salida." |
| `"<` | for French left double quotes (similar to <<). |
| `">` | for French right double quotes (similar to >>). |
| `\-` | like the old `\-`, but allowing hyphenation in the rest of the word. |

Table 8: The extra definitions made by `portuges.ldf`

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
34.1 ⟨*code⟩
34.2 \LdfInit\CurrentOption{captions\CurrentOption}
```

When this file is read as an option, i.e. by the `\usepackage` command, `portuges` will be an 'unknown' language in which case we have to make it known. So we check for the existence of `\l@portuges` to see whether we have to do something here. Since it is possible to load this file with any of the following four options to babel: portuges, portuguese, brazil and brazilian we also allow that the hyphenation patterns are loaded under any of these four names. We just have to find out which one was used.

```
34.3 \ifx\l@portuges\@undefined
34.4   \ifx\l@portuguese\@undefined
34.5     \ifx\l@brazil\@undefined
34.6       \ifx\l@brazilian\@undefined
34.7         \@nopatterns{Portuguese}
34.8         \adddialect\l@portuges0
34.9       \else
34.10        \let\l@portuges\l@brazilian
34.11      \fi
34.12    \else
34.13      \let\l@portuges\l@brazil
34.14    \fi
34.15  \else
34.16    \let\l@portuges\l@portuguese
```

[30]The file described in this section has version number v1.2p and was last revised on 2005/03/31. Contributions were made by Jose Pedro Ramalhete (`JRAMALHE@CERNVM` or `Jose-Pedro_Ramalhete@MACMAIL`) and Arnaldo Viegas de Lima `arnaldo@VNET.IBM.COM`.

```
  \fi
\fi
```

By now `\l@portuges` is defined. When the language definition file was loaded under a different name we make sure that the hyphenation patterns can be found.

```
\expandafter\ifx\csname l@\CurrentOption\endcsname\relax
  \expandafter\let\csname l@\CurrentOption\endcsname\l@portuges
\fi
```

Now we have to decide whether this language definition file was loaded for Portuguese or Brasilian use. This can be done by checking the contents of `\CurrentOption`. When it doesn't contain either 'portuges' or 'portuguese' we make `\bbl@tempb` empty.

```
\def\bbl@tempa{portuguese}
\ifx\CurrentOption\bbl@tempa
  \let\bbl@tempb\@empty
\else
  \def\bbl@tempa{portuges}
  \ifx\CurrentOption\bbl@tempa
    \let\bbl@tempb\@empty
  \else
    \def\bbl@tempb{brazil}
  \fi
\fi
\ifx\bbl@tempb\@empty
```

The next step consists of defining commands to switch to (and from) the Portuguese language.

`\captionsportuges` The macro `\captionsportuges` defines all strings used in the four standard documentclasses provided with LaTeX.

```
  \@namedef{captions\CurrentOption}{%
    \def\prefacename{Pref\'acio}%
    \def\refname{Refer\^encias}%
    \def\abstractname{Resumo}%
    \def\bibname{Bibliografia}%
    \def\chaptername{Cap\'{\i}tulo}%
    \def\appendixname{Ap\^endice}%
```

Some discussion took place around the correct translations for 'Table of Contents' and 'Index'. the translations differ for Portuguese and Brasilian based the following history:

> The whole issue is that some books without a real index at the end misused the term 'Índice' as table of contents. Then, what happens is that some books apeared with 'Índice' at the begining and a 'Índice Remissivo' at the end. Remissivo is a redundant word in this case, but was introduced to make up the difference. So in Brasil people started using 'Sumário' and 'Índice Remissivo'. In Portugal this seems not to be very common, therefore we chose 'Índice' instead of 'Índice Remissivo'.

```
  \def\contentsname{Conte\'udo}%
  \def\listfigurename{Lista de Figuras}%
  \def\listtablename{Lista de Tabelas}%
  \def\indexname{\'Indice}%
  \def\figurename{Figura}%
  \def\tablename{Tabela}%
  \def\partname{Parte}%
  \def\enclname{Anexo}%
  \def\ccname{Com c\'opia a}%
  \def\headtoname{Para}%
  \def\pagename{P\'agina}%
  \def\seename{ver}%
  \def\alsoname{ver tamb\'em}%
```

An alternate term for ‘Proof’ could be ‘Prova’.

```
  \def\proofname{Demonstra\c{c}\~ao}%
  \def\glossaryname{Gloss\'ario}%
  }
```

`\dateportuges` The macro `\dateportuges` redefines the command `\today` to produce Portuguese dates.

```
 \@namedef{date\CurrentOption}{%
  \def\today{\number\day\space de\space\ifcase\month\or
   Janeiro\or Fevereiro\or Mar\c{c}o\or Abril\or Maio\or Junho\or
   Julho\or Agosto\or Setembro\or Outubro\or Novembro\or Dezembro%
   \fi
   \space de\space\number\year}}
\else
```

For the Brasilian version of these definitions we just add a “dialect”.

```
 \expandafter
  \adddialect\csname l@\CurrentOption\endcsname\l@portuges
```

`\captionsbrazil` The “captions” are different for both versions of the language, so we define the macro `\captionsbrazil` here.

```
 \@namedef{captions\CurrentOption}{%
  \def\prefacename{Pref\'acio}%
  \def\refname{Refer\^encias}%
  \def\abstractname{Resumo}%
  \def\bibname{Refer\^encias Bibliogr\'aficas}%
  \def\chaptername{Cap\'{\i}tulo}%
  \def\appendixname{Ap\^endice}%
  \def\contentsname{Sum\'ario}%
  \def\listfigurename{Lista de Figuras}%
  \def\listtablename{Lista de Tabelas}%
  \def\indexname{\'Indice Remissivo}%
  \def\figurename{Figura}%
  \def\tablename{Tabela}%
  \def\partname{Parte}%
```

```
    \def\enclname{Anexo}%
    \def\ccname{C\'opia para}%
    \def\headtoname{Para}%
    \def\pagename{P\'agina}%
    \def\seename{veja}%
    \def\alsoname{veja tamb\'em}%
    \def\proofname{Demonstra\c{c}\~ao}%
    \def\glossaryname{Glossary}% <-- Needs translation
    }
```

`\datebrazil` The macro `\datebrazil` redefines the command `\today` to produce Brasilian dates, for which the names of the months are not capitalized.

```
  \@namedef{date\CurrentOption}{%
    \def\today{\number\day\space de\space\ifcase\month\or
      janeiro\or fevereiro\or mar\c{c}o\or abril\or maio\or junho\or
      julho\or agosto\or setembro\or outubro\or novembro\or dezembro%
      \fi
      \space de\space\number\year}}
\fi
```

`\portugeshyphenmins` Set correct values for `\lefthyphenmin` and `\righthyphenmin`.

```
\providehyphenmins{\CurrentOption}{\tw@\thr@@}
```

`\extrasportuges` `\noextrasportuges` The macro `\extrasportuges` will perform all the extra definitions needed for the Portuguese language. The macro `\noextrasportuges` is used to cancel the actions of `\extrasportuges`.

For Portuguese the `"` character is made active. This is done once, later on its definition may vary. Other languages in the same document may also use the `"` character for shorthands; we specify that the portuguese group of shorthands should be used.

```
\initiate@active@char{"}
\@namedef{extras\CurrentOption}{\languageshorthands{portuges}}
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \bbl@activate{"}}
```

Don't forget to turn the shorthands off again.

```
\addto\noextrasportuges{\bbl@deactivate{"}}
```

First we define access to the guillemets for quotations,

```
\declare@shorthand{portuges}{"<}{%
  \textormath{\guillemotleft}{\mbox{\guillemotleft}}}
\declare@shorthand{portuges}{">}{%
  \textormath{\guillemotright}{\mbox{\guillemotright}}}
```

then we define two shorthands to be able to specify hyphenation breakpoints that behave a little different from `\-`.

```
\declare@shorthand{portuges}{"-}{\nobreak-\bbl@allowhyphens}
\declare@shorthand{portuges}{""}{\hskip\z@skip}
```

And we want to have a shorthand for disabling a ligature.

```
\declare@shorthand{portuges}{"|}{%
  \textormath{\discretionary{-}{}{\kern.03em}}{}}
```

\- All that is left now is the redefinition of `\-`. The new version of `\-` should indicate an extra hyphenation position, while allowing other hyphenation positions to be generated automatically. The standard behaviour of TEX in this respect is very unfortunate for languages such as Dutch and German, where long compound words are quite normal and all one needs is a means to indicate an extra hyphenation position on top of the ones that TEX can generate from the hyphenation patterns.

```
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \babel@save\-}
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \def\-{\allowhyphens\discretionary{-}{}{}\allowhyphens}}
```

\ord \ro \orda \ra We also provide an easy way to typeset ordinals, both in the male (`\ord` or `\ro`) and the female (`orda` or `\ra`) form.

```
\def\ord{$^{\rm o}$}
\def\orda{$^{\rm a}$}
\let\ro\ord\let\ra\orda
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish\CurrentOption
⟨/code⟩
```

## 35 The Spanish language

The file `spanish.dtx`[31] defines all the language-specific macros for the Spanish language.

Custumization is made following mainly the books on the subject by José Martínez de Sousa. By typesetting `spanish.dtx` directly you will get the full documentation (regrettably is in Spanish only, but it is pretty long). References in this part refers to that document. There are several aditional features documented in the Spanish version only.

This style provides:

- Translations following the International LaTeX conventions, as well as `\today`.
- Shorthands listed in Table 9. Examples in subsection 3.4 are illustrative. Note that `"~` has a special meaning in `spanish` different to other languages, and is used mainly in linguistic contexts.

| | |
|---|---|
| `'a` | acute accented a. Also for: e, i, o, u (both lowercase and uppercase). |
| `'n` | ñ (also uppercase). |
| `~n` | ñ (also uppercase). Deprecated. |
| `"u` | ü (also uppercase). |
| `"a` | Ordinal numbers (also `"A`, `"o`, `"O`). |
| `"c` | ç (also uppercase). |
| `"rr` | rr, but -r when hyphenated |
| `"-` | Like `\-`, but allowing hyphenation in the rest the word. |
| `"=` | Like `-`, but allowing hyphenation in the rest the word. |
| `"~` | The hyphen is repeated at the very beginning of the next line if the word is hyphenated at this point. |
| `""` | Like `"-` but producing no hyphen sign. |
| `~-` | Like - but with no break after the hyphen. Also for: en-dashes (`~--`) and em-dashes (`~---`). |
| `"/` | A slash slightly lowered, if necessary. |
| `"\|` | disable ligatures at this point. |
| `"<` | Left guillemets. |
| `">` | Right guillemets. |
| `<<` | `\begin{quoting}`. (See text.) |
| `>>` | `\end{quoting}`. (See text.) |

Table 9: Extra definitions made by file `spanish.ldf`

[31]The file described in this section has version number v4.2b and was last revised on 2004/02/20. The original author from v4.0 on is Javier Bezos. Previous versions were by Julio Sánchez.

- `\deactivatetilden` deactivates the `~n` and `~N` shorthands.
- *In math mode* a dot followed by a digit is replaced by a decimal comma.
- Spanish ordinals and abbeviations with `\sptext` as, for instance, `1\sptext{er}`. The preceptive dot is included.
- Accented functions: lím, máx, mín, mód. You may globally omit the accents with `\unaccentedoperators`. Spaced functions: arc cos, etc. You may globally kill that space with `\unspacedoperators`. `\dotlessi` is provided for use in math mode.
- A `quoting` environment and a related pair of shorthands `<<` and `>>`. The command `\deactivatequoting` deactivates these shorthand in case you want to use `<` and `>` in some AMS commands and numerical comparisons.
- The command `\selectspanish` selects the `spanish` language *and* its shorthands. (Intended for the preamble.)
- `\frenchspacing` is used.
- Ellipsis are best typed `...` or, within a sentence, `\...`
- There is a small space before `\%`.
- `\lsc` provides lowercase small caps. (See subsection 3.10.)

Just in case `spanish` is the main language, the group `\layoutspanish` is activated, which modifies the standard classes through the whole document (it cannot be deactivated) in the following way:

- Both `enumerate` and `itemize` are adapted to Spanish rules.
- Both `\alph` and `\Alph` include *ñ* after *n*.
- Symbol footmarks are one, two, three, etc., asteriscs.
- `OT1` guillemets are generated with two `lasy` symbols instead of small `\ll` and `\gg`.
- `\roman` is redefined to write small caps roman numerals, since lowercase roman numerals are not allowed. However, *MakeIndex* rejects entries containing pages in that format. The `.idx` file must be preprocessed if the document has this kind of entries with the provided `romanidx.tex` tool—just TeX it and follow the instructions.
- There is a dot after section numbers in titles and toc.

This group is ignored if you write `\selectspanish*` in the preamble.

Some additional commands are provided to be used in the `spanish.cfg` file:

- With `\es@activeacute` acute accents are always active, overriding the default babel behaviour.

- `\es@enumerate` sets the labels to be used by `enumerate`. The same applies to `\es@itemize` and `itemize`.
- `\es@operators` stores the operator commands. All of them are canceled with

  ```
  \let\es@operators\relax
  ```

The commands `\deactivatequoting`, `\deactivatetilden` and `\selectspanish` may be used in this file, too.

A subset of these commands is provided for use in Plain TEX (with `\input spanish.sty`).

## 35.1 The Code

This file provides definition for both LaTeX 2ε and non LaTeX 2ε formats.

Identify the `ldf` file.

```
⟨*code⟩
\ProvidesLanguage{spanish.ldf}
        [2005/03/31 v4.2b Spanish support from the babel system]
```

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc. When this file is read as an option, i.e. by the `\usepackage` command, `spanish` will be an 'unknown' language in which case we have to make it known. So we check for the existence of `\l@spanish` to see whether we have to do something here.

```
\LdfInit{spanish}\captionsspanish
\ifx\undefined\l@spanish
  \@nopatterns{Spanish}
  \adddialect\l@spanish0
\fi
```

We define some tools which will be used in that style file: (1) we make sure that ~ is active, (2) `\es@delayed` delays the expansion of the code in conditionals (in fact, quite similar to `\bbl@afterfi`).

```
\edef\es@savedcatcodes{%
  \catcode`\noexpand\~=\the\catcode`\~
  \catcode`\noexpand\"=\the\catcode`\"}
\catcode`\~=\active
\catcode`\"=12
\long\def\es@delayed#1\then#2\else#3\fi{%
  #1%
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi
  {#2}{#3}}
```

Two tests are introduced. The first one tells us if the format is LaTeX 2ε, and the second one if the format is plain or any other. If both are false, the format is LaTeX2.09.

```
\es@delayed
\expandafter\ifx\csname documentclass\endcsname\relax\then
  \let\ifes@LaTeXe\iffalse
\else
  \let\ifes@LaTeXe\iftrue
\fi
\es@delayed
\expandafter\ifx\csname newenvironment\endcsname\relax\then
  \let\ifes@plain\iftrue
\else
  \let\ifes@plain\iffalse
\fi
```

Translations for captions.

```
\addto\captionsspanish{%
  \def\prefacename{Prefacio}%
  \def\refname{Referencias}%
  \def\abstractname{Resumen}%
  \def\bibname{Bibliograf\'{\i}a}%
  \def\chaptername{Cap\'{\i}tulo}%
  \def\appendixname{Ap\'endice}%
  \def\listfigurename{\'Indice de figuras}%
  \def\listtablename{\'Indice de cuadros}%
  \def\indexname{\'Indice alfab\'etico}%
  \def\figurename{Figura}%
  \def\tablename{Cuadro}%
  \def\partname{Parte}%
  \def\enclname{Adjunto}%
  \def\ccname{Copia a}%
  \def\headtoname{A}%
  \def\pagename{P\'agina}%
  \def\seename{v\'ease}%
  \def\alsoname{v\'ease tambi\'en}%
  \def\proofname{Demostraci\'on}%
  \def\glossaryname{Glosario}}

\expandafter\ifx\csname chapter\endcsname\relax
  \addto\captionsspanish{\def\contentsname{\'Indice}}
\else
  \addto\captionsspanish{\def\contentsname{\'Indice general}}
\fi
```

And the date.

```
\def\datespanish{%
 \def\today{\the\day~de \ifcase\month\or enero\or febrero\or
      marzo\or abril\or mayo\or junio\or julio\or agosto\or
      septiembre\or octubre\or noviembre\or diciembre\fi
      \ de\ifnum\year>1999\es@yearl\fi~\the\year}}
\def\spanishdatedel{\def\es@yearl{l}}
\def\spanishdatede{\let\es@yearl\@empty}
\spanishdatede
```

The basic macros to select the language, in the preamble or the config file. Use of `\selectlanguage` should be avoided at this early stage because the active chars are not yet active. `\selectspanish` makes them active.

```
\def\selectspanish{%
  \def\selectspanish{%
    \def\selectspanish{%
      \PackageWarning{spanish}{Extra \string\selectspanish ignored}}%
    \es@select}}

\@onlypreamble\selectspanish

\def\es@select{%
  \let\es@select\@undefined
  \selectlanguage{spanish}%
  \catcode`\"\active\catcode`\~=\active}
```

Instead of joining all the extras directly in `\extrasspanish`, we subdivide them in three further groups.

```
\def\extrasspanish{%
  \textspanish
  \mathspanish
  \ifx\shorthandsspanish\@empty
    \spanishdeactivate{."'~<>}%
    \languageshorthands{none}%
  \else
    \shorthandsspanish
  \fi}
\def\noextrasspanish{%
  \ifx\textspanish\@empty\else
    \notextspanish
  \fi
  \ifx\mathspanish\@empty\else
    \nomathspanish
  \fi
  \ifx\shorthandsspanish\@empty\else
    \noshorthandsspanish
  \fi
  \es@reviveshorthands}
```

And the first of these sub-groups is defined.

```
\addto\textspanish{%
  \babel@save\sptext
  \def\sptext{\protect\es@sptext}}
```

The definition of `\sptext` is more elaborated than that of `\textsuperscript`. With uppercase superscript text the scriptscriptsize is used. The mandatory dot is already included. There are two versions, depending on the format.

```
\ifes@LaTeXe   %<<<<<<
  \newcommand\es@sptext[1]{%
    {.\setbox\z@\hbox{8}\dimen@\ht\z@
     \csname S@\f@size\endcsname
    \edef\@tempa{\def\noexpand\@tempc{#1}%
      \lowercase{\def\noexpand\@tempb{#1}}}\@tempa
    \ifx\@tempb\@tempc
      \fontsize\sf@size\z@
      \selectfont
      \advance\dimen@-1.15ex
    \else
      \fontsize\ssf@size\z@
      \selectfont
      \advance\dimen@-1.5ex
    \fi
    \math@fontsfalse\raise\dimen@\hbox{#1}}}
\else          %<<<<<<
  \let\sptextfont\rm
  \newcommand\es@sptext[1]{%
    {.\setbox\z@\hbox{8}\dimen@\ht\z@
    \edef\@tempa{\def\noexpand\@tempc{#1}%
      \lowercase{\def\noexpand\@tempb{#1}}}\@tempa
    \ifx\@tempb\@tempc
      \advance\dimen@-0.75ex
      \raise\dimen@\hbox{$\scriptstyle\sptextfont#1$}%
    \else
      \advance\dimen@-0.8ex
      \raise\dimen@\hbox{$\scriptscriptstyle\sptextfont#1$}%
    \fi}}
\fi            %<<<<<<
```

Now, lowercase small caps. First, we test if there are actual small caps for the current font. If not, faked small caps are used. The `\selectfont` in `\es@lsc` could seem redundant, but it's not.

```
\ifes@LaTeXe   %<<<<<<
  \addto\textspanish{%
    \babel@save\lsc
    \def\lsc{\protect\es@lsc}}

  \def\es@lsc#1{%
    \leavevmode
    \hbox{\scshape\selectfont
      \expandafter\ifx\csname\f@encoding/\f@family/\f@series
          /n/\f@size\expandafter\endcsname
        \csname\curr@fontshape/\f@size\endcsname
        \csname S@\f@size\endcsname
        \fontsize\sf@size\z@\selectfont
          \PackageInfo{spanish}{Replacing undefined sc font\MessageBreak
                                shape by faked small caps}%
        \MakeUppercase{#1}%
      \else
        \MakeLowercase{#1}%
      \fi}}
\fi            %<<<<<<
```

The `quoting` environment. This part is not available in Plain, hence the test. Overriding the default `\everypar` is a bit tricky.

```
\newif\ifes@listquot

\ifes@plain\else %<<<<<<
  \csname newtoks\endcsname\es@quottoks
  \csname newcount\endcsname\es@quotdepth

  \newenvironment{quoting}
   {\leavevmode
    \advance\es@quotdepth1
    \csname lquot\romannumeral\es@quotdepth\endcsname%
    \ifnum\es@quotdepth=\@ne
      \es@listquotfalse
      \let\es@quotpar\everypar
      \let\everypar\es@quottoks
      \everypar\expandafter{\the\es@quotpar}%
      \es@quotpar{\the\everypar
        \ifes@listquot\global\es@listquotfalse\else\es@quotcont\fi}%
    \fi
    \toks@\expandafter{\es@quotcont}%
    \edef\es@quotcont{\the\toks@
       \expandafter\noexpand
       \csname rquot\romannumeral\es@quotdepth\endcsname}}
   {\csname rquot\romannumeral\es@quotdepth\endcsname}

  \def\lquoti{\guillemotleft{}}
  \def\rquoti{\guillemotright{}}
  \def\lquotii{``}
  \def\rquotii{''}
  \def\lquotiii{`}
  \def\rquotiii{'}

  \let\es@quotcont\@empty
```

If there is a margin par inside quoting, we don't add the quotes. `\es@listqout` stores the quotes to be used before item labels; otherwise they could appear after the labels.

```
  \addto\@marginparreset{\let\es@quotcont\@empty}

  \def\es@listquot{%
    \csname rquot\romannumeral\es@quotdepth\endcsname
    \global\es@listquottrue}
\fi            %<<<<<<
```

Now, the `\frenchspacing`, followed by `\...` and `\%`

```
\addto\textspanish{\bbl@frenchspacing}
\addto\notextspanish{\bbl@nonfrenchspacing}

\addto\textspanish{%
  \let\es@save@dot\.%
  \babel@save\.%
  \def\.{\@ifnextchar.{\es@dots}{\es@save@dot}}}

\def\es@dots..{\leavevmode\hbox{...}\spacefactor\@M}

\ifes@LaTeXe   %<<<<<<
  \addto\textspanish{%
    \let\percentsign\%%
    \babel@save\%%
    \def\%{\unskip\,\percentsign{}}}
\else
  \addto\textspanish{%
    \let\percentsign\%%
    \babel@save\%%
    \def\%{\unskip\ifmmode\,\else$\m@th\,$\fi\percentsign{}}}
\fi
```

We follow with the math group. It's not easy to add an accent in an operator. The difficulty is that we must avoid using text (that is, `\mbox`) because we have no control on font and size, and at time we should access `\i`, which is a text command forbidden in math mode. `\dotlessi` must be converted to uppercase if necessary in LaTeX $2_\varepsilon$. There are two versions, depending on the format.

```
\addto\mathspanish{%
  \babel@save\dotlessi
  \def\dotlessi{\protect\es@dotlessi}}

\let\nomathspanish\relax %% Unused, but called

\ifes@LaTeXe   %<<<<<<
  \def\es@texti{\i}
  \addto\@uclclist{\dotlessi\es@texti}
\fi            %<<<<<<

\ifes@LaTeXe   %<<<<<<
  \def\es@dotlessi{%
    \ifmmode
      {\ifnum\mathgroup=\m@ne
         \imath
       \else
         \count@\escapechar \escapechar=\m@ne
         \expandafter\expandafter\expandafter
           \split@name\expandafter\string\the\textfont\mathgroup\@nil
         \escapechar=\count@
         \@ifundefined{\f@encoding\string\i}%
           {\edef\f@encoding{\string?}}{}%
         \expandafter\count@\the\csname\f@encoding\string\i\endcsname
         \advance\count@"7000
         \mathchar\count@
       \fi}%
    \else
```

```
      \i
    \fi}
\else          %<<<<<<
  \def\es@dotlessi{%
    \ifmmode
      \mathchar"7010
    \else
      \i
    \fi}
\fi            %<<<<<<
```

The switches for accents and spaces in math.

```
\def\accentedoperators{%
  \def\es@op@ac##1{\acute{##1}}%
  \def\es@op@i{\acute{\dotlessi}}}
\def\unaccentedoperators{%
  \def\es@op@ac##1{##1}%
  \def\es@op@i{i}}
\accentedoperators

\def\spacedoperators{\let\es@op@sp\,}
\def\unspacedoperators{\let\es@op@sp\@empty}
\spacedoperators
```

The operators are stored in `\es@operators`, which in turn is included in the math group. Since `\operator@font` is defined in LaTeX 2$_\varepsilon$ only, we defined in the plain variant.

```
\addto\mathspanish{\es@operators}

\ifes@LaTeXe\else %<<<<<<
  \let\operator@font\rm
  \def\@empty{}
\fi            %<<<<<<

\def\es@operators{%
  \babel@save\lim
  \def\lim{\mathop{\operator@font l\protect\es@op@i m}}%
  \babel@save\limsup
  \def\limsup{\mathop{\operator@font l\es@op@i m\,sup}}%
  \babel@save\liminf
  \def\liminf{\mathop{\operator@font l\es@op@i m\,inf}}%
  \babel@save\max
  \def\max{\mathop{\operator@font m\es@op@ac ax}}%
  \babel@save\inf
  \def\inf{\mathop{\operator@font \protect\es@op@i nf}}%
  \babel@save\min
  \def\min{\mathop{\operator@font m\protect\es@op@i n}}%
  \babel@save\bmod
  \def\bmod{%
    \nonscript\mskip-\medmuskip\mkern5mu%
    \mathbin{\operator@font m\es@op@ac od}\penalty900\mkern5mu%
    \nonscript\mskip-\medmuskip}%
  \babel@save\pmod
  \def\pmod##1{%
    \allowbreak\mkern18mu({\operator@font m\es@op@ac od}\,\,##1)}%
  \def\es@a##1 {%
    \es@delayed
    \if^##1^\then  %  is it empty? do nothing and continue
      \es@a
    \else
      \es@delayed
      \if&##1\then % is it &? do nothing and finish
      \else
        \begingroup
          \let\,\@empty % \, is ignored when def'ing the macro name
          \let\acute\@firstofone % same
          \edef\es@b{\expandafter\noexpand\csname##1\endcsname}%
          \def\,{\noexpand\es@op@sp}%
          \def\acute####1{%
            \if i####1%
              \noexpand\es@op@i
            \else
              \noexpand\es@op@ac####1%
            \fi}%
          \edef\es@a{\endgroup
            \noexpand\babel@save\expandafter\noexpand\es@b
            \def\expandafter\noexpand\es@b{%
                    \mathop{\noexpand\operator@font##1}\nolimits}}%
          \es@a % It restores itself
        \es@a
      \fi
    \fi}%
  \let\es@b\spanishoperators
  \addto\es@b{ }%
  \expandafter\es@a\es@b sen tg arc\,sen arc\,cos arc\,tg & }

\def\spanishoperators{cotg cosec senh tgh}
```

Now comes the text shorthands. They are grouped in `\shorthandsspanish` and this style performs some operations before the babel shortands are called. The goals are to allow espression like `$a^{x'}$` and to deactivate the shorthands making them of category 'other.' After providing a `\'i` shorthand, the new macros are defined.

```
\DeclareTextCompositeCommand{\'}{OT1}{i}{\@tabacckludge'{\i}}

\def\es@set@shorthand#1{%
  \expandafter\edef\csname es@savecat\string#1\endcsname
     {\the\catcode`#1}%
  \initiate@active@char{#1}%
  \catcode`#1=\csname es@savecat\string#1\endcsname\relax
  \expandafter\let\csname es@math\string#1\expandafter\endcsname
```

```
    \csname normal@char\string#1\endcsname}

\def\es@use@shorthand{%
  \es@delayed
  \ifx\thepage\relax\then
    \string
  \else{%
    \es@delayed
    \ifx\protect\@unexpandable@protect\then
      \noexpand
    \else
      \es@use@sh
    \fi}%
  \fi}

\def\es@text@sh#1{\csname active@char\string#1\endcsname}
\def\es@math@sh#1{\csname es@math\string#1\endcsname}

\def\es@use@sh{%
  \es@delayed
  \if@safe@actives\then
    \string
  \else{%
    \es@delayed
    \ifmmode\then
      \es@math@sh
    \else
      \es@text@sh
    \fi}%
  \fi}

\gdef\es@activate#1{%
  \begingroup
    \lccode`\~=`#1
    \lowercase{%
  \endgroup
  \def~{\es@use@shorthand~}}}

\def\spanishdeactivate#1{%
  \@tfor\@tempa:=#1\do{\expandafter\es@spdeactivate\@tempa}}

\def\es@spdeactivate#1{%
  \if.#1%
    \mathcode`\.=\es@period@code
  \else
    \begingroup
      \lccode`\~=`#1
      \lowercase{%
    \endgroup
    \expandafter\let\expandafter~%
    \csname normal@char\string#1\endcsname}%
   \catcode`#1\csname es@savecat\string#1\endcsname\relax
  \fi}

\def\es@reviveshorthands{%
  \es@restore{"}\es@restore{~}%
  \es@restore{<}\es@restore{>}%
  \es@quoting}

\def\es@restore#1{%
  \catcode`#1=\active
  \begingroup
    \lccode`\~=`#1
    \lowercase{%
  \endgroup
  \bbl@deactivate{~}}}
```

But spanish allows two category codes for ', so both should be taken into account in `\bbl@pr@m@s`.

```
\begingroup
\catcode`\'=12
\lccode`~=`' \lccode`'=`'
\lowercase{%
\gdef\bbl@pr@m@s{%
  \es@delayed
  \ifx~\@let@token\then
    \pr@@@s
  \else
    {\es@delayed
     \ifx'\@let@token\then
       \pr@@@s
     \else
       {\es@delayed
        \ifx^\@let@token\then
          \pr@@@t
        \else
          \egroup
        \fi}%
     \fi}%
  \fi}}
\endgroup
```

```
\expandafter\ifx\csname @tabacckludge\endcsname\relax
  \let\es@tak\a
\else
  \let\es@tak\@tabacckludge
\fi

\ifes@LaTeXe   %<<<<<<
  \def\@tabacckludge#1{\expandafter\es@tak\string#1}
  \let\a\@tabacckludge
```

```
\else\ifes@plain %<<<<<<
  \def\@tabacckludge#1{\csname\string#1\endcsname}
\else          %<<<<<<
  \def\@tabacckludge#1{\csname a\string#1\endcsname}
\fi\fi         %<<<<<<

\expandafter\ifx\csname add@accent\endcsname\relax
  \def\add@accent#1#2{\accent#1 #2}
\fi
```

Instead of redefining `\'`, we redefine the internal macro for the OT1 encoding.

```
\ifes@LaTeXe   %<<<<<<
  \def\es@accent#1#2#3{%
    \expandafter\@text@composite
    \csname OT1\string#1\endcsname#3\@empty\@text@composite
    {\bbl@allowhyphens\add@accent{#2}{#3}\bbl@allowhyphens
     \setbox\@tempboxa\hbox{#3%
       \global\mathchardef\accent@spacefactor\spacefactor}%
     \spacefactor\accent@spacefactor}}
\else          %<<<<<<
  \def\es@accent#1#2#3{%
    \bbl@allowhyphens\add@accent{#2}{#3}\bbl@allowhyphens
    \spacefactor\sfcode`#3 }
\fi            %<<<<<<
```

The shorthands are activated in the aux file. Now, we begin the shorthands group.

```
\addto\shorthandsspanish{\languageshorthands{spanish}}
\let\noshorthandsspanish\relax
```

First, decimal comma.

```
\def\spanishdecimal#1{\def\es@decimal{{#1}}}
\def\decimalcomma{\spanishdecimal{,}}
\def\decimalpoint{\spanishdecimal{.}}
\decimalcomma

\es@set@shorthand{.}

\@namedef{es@math\string.}{%
  \@ifnextchar\egroup
    {\mathchar\es@period@code\relax}%
    {\es@text@sh.}}

\declare@shorthand{system}{.}{\mathchar\es@period@code\relax}
\addto\shorthandsspanish{%
  \mathchardef\es@period@code\the\mathcode`\.%
  \babel@savevariable{\mathcode`\.}%
  \mathcode`\.="8000 %
  \es@activate{.}}

```

```
\AtBeginDocument{%
  \catcode`\.=12
  \if@filesw
    \immediate\write\@mainaux{%
    \string\catcode`\string\.=12}%
  \fi}

\declare@shorthand{spanish}{.1}{\es@decimal1}
\declare@shorthand{spanish}{.2}{\es@decimal2}
\declare@shorthand{spanish}{.3}{\es@decimal3}
\declare@shorthand{spanish}{.4}{\es@decimal4}
\declare@shorthand{spanish}{.5}{\es@decimal5}
\declare@shorthand{spanish}{.6}{\es@decimal6}
\declare@shorthand{spanish}{.7}{\es@decimal7}
\declare@shorthand{spanish}{.8}{\es@decimal8}
\declare@shorthand{spanish}{.9}{\es@decimal9}
\declare@shorthand{spanish}{.0}{\es@decimal0}
```

Now accents and tools

```
\es@set@shorthand{"}
\def\es@umlaut#1{%
  \bbl@allowhyphens\add@accent{127}#1\bbl@allowhyphens
  \spacefactor\sfcode`#1 }
```

We override the default " of babel, intended for german.

```
\ifes@LaTeXe   %<<<<<<
  \addto\shorthandsspanish{%
    \es@activate{"}%
    \es@activate{~}%
    \babel@save\bbl@umlauta
    \let\bbl@umlauta\es@umlaut
    \expandafter\babel@save\csname OT1\string\~\endcsname
    \expandafter\def\csname OT1\string\~\endcsname{\es@accent\~{126}}%
    \expandafter\babel@save\csname OT1\string\'\endcsname
    \expandafter\def\csname OT1\string\'\endcsname{\es@accent\'{19}}}
\else          %<<<<<<
  \addto\shorthandsspanish{%
    \es@activate{"}%
    \es@activate{~}%
    \babel@save\bbl@umlauta
    \let\bbl@umlauta\es@umlaut
    \babel@save\~%
    \def\~{\es@accent\~{126}}%
    \babel@save\'%
    \def\'#1{\if#1i\es@accent\'{19}\i\else\es@accent\'{19}{#1}\fi}}
\fi            %<<<<<<

\declare@shorthand{spanish}{"a}{\protect\es@sptext{a}}
\declare@shorthand{spanish}{"A}{\protect\es@sptext{A}}
\declare@shorthand{spanish}{"o}{\protect\es@sptext{o}}
\declare@shorthand{spanish}{"O}{\protect\es@sptext{O}}
```

```
\declare@shorthand{spanish}{"e}{\protect\es@sptext@r{e}}
\declare@shorthand{spanish}{"E}{\protect\es@sptext@r{E}}

\def\es@sptext@r#1#2{\es@sptext{#1#2}}

\declare@shorthand{spanish}{"u}{\"u}
\declare@shorthand{spanish}{"U}{\"U}

\declare@shorthand{spanish}{"c}{\c{c}}
\declare@shorthand{spanish}{"C}{\c{C}}

\declare@shorthand{spanish}{"<}{\guillemotleft{}}
\declare@shorthand{spanish}{">}{\guillemotright{}}
\declare@shorthand{spanish}{"-}{\bbl@allowhyphens\-\bbl@allowhyphens}
\declare@shorthand{spanish}{"=}%
  {\bbl@allowhyphens\char\hyphenchar\font\hskip\z@skip}
\declare@shorthand{spanish}{"~}
  {\bbl@allowhyphens\discretionary{\char\hyphenchar\font}%
       {\char\hyphenchar\font}{\char\hyphenchar\font}\bbl@allowhyphens}
\declare@shorthand{spanish}{"r}
  {\bbl@allowhyphens\discretionary{\char\hyphenchar\font}%
       {}{r}\bbl@allowhyphens}
\declare@shorthand{spanish}{"R}
  {\bbl@allowhyphens\discretionary{\char\hyphenchar\font}%
       {}{R}\bbl@allowhyphens}
\declare@shorthand{spanish}{"y}
  {\@ifundefined{scalebox}%
     {\ensuremath{\tau}}%
     {\raisebox{1ex}{\scalebox{-1}{\resizebox{.45em}{1ex}{2}}}}}
\declare@shorthand{spanish}{""}{\hskip\z@skip}
\declare@shorthand{spanish}{"/}
  {\setbox\z@\hbox{/}%
   \dimen@\ht\z@
   \advance\dimen@-1ex
   \advance\dimen@\dp\z@
   \dimen@.31\dimen@
   \advance\dimen@-\dp\z@
   \ifdim\dimen@>0pt
     \kern.01em\lower\dimen@\box\z@\kern.03em
   \else
     \box\z@
   \fi}
\declare@shorthand{spanish}{"?}
  {\setbox\z@\hbox{?`}%
   \leavevmode\raise\dp\z@\box\z@}
\declare@shorthand{spanish}{"!}
  {\setbox\z@\hbox{!`}%
   \leavevmode\raise\dp\z@\box\z@}

\es@set@shorthand{~}
```

```
\declare@shorthand{spanish}{~n}{\~n}
\declare@shorthand{spanish}{~N}{\~N}
\declare@shorthand{spanish}{~-}{%
  \leavevmode
  \bgroup
  \let\@sptoken\es@dashes  % This assignation changes the
  \@ifnextchar-%             \@ifnextchar behaviour
    {\es@dashes}%
    {\hbox{\char\hyphenchar\font}\egroup}}
\def\es@dashes-{%
  \@ifnextchar-%
    {\bbl@allowhyphens\hbox{---}\bbl@allowhyphens\egroup\@gobble}%
    {\bbl@allowhyphens\hbox{--}\bbl@allowhyphens\egroup}}

\def\deactivatetilden{%
  \expandafter\let\csname spanish@sh@\string~@n@\endcsname\relax
  \expandafter\let\csname spanish@sh@\string~@N@\endcsname\relax}
```

The shorthands for `quoting`.

```
\expandafter\ifx\csname XML@catcodes\endcsname\relax
  \addto\es@select{%
    \catcode`\<\active\catcode`\>=\active
    \es@quoting}

  \es@set@shorthand{<}
  \es@set@shorthand{>}

  \declare@shorthand{system}{<}{\csname normal@char\string<\endcsname}
  \declare@shorthand{system}{>}{\csname normal@char\string>\endcsname}

  \addto\shorthandsspanish{%
    \es@activate{<}%
    \es@activate{>}}
  \ifes@LaTeXe   %<<<<<<
    \AtBeginDocument{%
      \es@quoting
      \if@filesw
        \immediate\write\@mainaux{\string\es@quoting}%
      \fi}%
  \fi            %<<<<<<

  \def\activatequoting{%
    \catcode`>=\active \catcode`<=\active
    \let\es@quoting\activatequoting}
  \def\deactivatequoting{%
    \catcode`>=12 \catcode`<=12
    \let\es@quoting\deactivatequoting}

  \declare@shorthand{spanish}{<<}{\begin{quoting}}
  \declare@shorthand{spanish}{>>}{\end{quoting}}
```

```
\fi

\let\es@quoting\relax
\let\deactivatequoting\relax
\let\activatequoting\relax
```

The acute accents are stored in a macro. If `activeacute` was set as an option it's executed. If not is not deleted for a possible later use in the `cfg` file. In non LaTeX $2_\varepsilon$ formats is always executed.

```
\def\es@activeacute{%
  \es@set@shorthand{'}%
  \addto\shorthandsspanish{\es@activate{'}}%
  \addto\es@reviveshorthands{\es@restore{'}}%
  \addto\es@select{\catcode`'=\active}%
  \declare@shorthand{spanish}{'a}{\@tabacckludge'a}%
  \declare@shorthand{spanish}{'A}{\@tabacckludge'A}%
  \declare@shorthand{spanish}{'e}{\@tabacckludge'e}%
  \declare@shorthand{spanish}{'E}{\@tabacckludge'E}%
  \declare@shorthand{spanish}{'i}{\@tabacckludge'i}%
  \declare@shorthand{spanish}{'I}{\@tabacckludge'I}%
  \declare@shorthand{spanish}{'o}{\@tabacckludge'o}%
  \declare@shorthand{spanish}{'O}{\@tabacckludge'O}%
  \declare@shorthand{spanish}{'u}{\@tabacckludge'u}%
  \declare@shorthand{spanish}{'U}{\@tabacckludge'U}%
  \declare@shorthand{spanish}{'n}{\~n}%
  \declare@shorthand{spanish}{'N}{\~N}%
  \declare@shorthand{spanish}{''}{\textquotedblright}%
  \let\es@activeacute\relax}

\ifes@LaTeXe   %<<<<<<
  \@ifpackagewith{babel}{activeacute}{\es@activeacute}{}
\else          %<<<<<<
  \es@activeacute
\fi            %<<<<<<%
```

And the customization. By default these macros only store the values and do nothing.

```
\def\es@enumerate#1#2#3#4{%
  \def\es@enum{{#1}{#2}{#3}{#4}}}

\def\es@itemize#1#2#3#4{%
  \def\es@item{{#1}{#2}{#3}{#4}}}
```

The part formerly in the `.lld` file comes here. It performs layout adaptation of LaTeX to “orthodox” Spanish rules.

```
\ifes@LaTeXe   %<<<<<<

\es@enumerate{1.}{a)}{1)}{a$'$}
\def\spanishdashitems{\es@itemize{---}{---}{---}{---}}
\def\spanishsymbitems{%
  \es@itemize
```

```
    {\leavevmode\hbox to 1.2ex
      {\hss\vrule height .9ex width .7ex depth -.2ex\hss}}%
    {\textbullet}%
    {$\m@th\circ$}%
    {$\m@th\diamond$}}
\def\spanishsignitems{%
  \es@itemize
    {\textbullet}%
    {$\m@th\circ$}%
    {$\m@th\diamond$}%
    {$\m@th\triangleright$}}
\spanishsymbitems

\def\es@enumdef#1#2#3\@@{%
  \if#21%
    \@namedef{theenum#1}{\arabic{enum#1}}%
  \else\if#2a%
    \@namedef{theenum#1}{\emph{\alph{enum#1}}}%
  \else\if#2A%
    \@namedef{theenum#1}{\Alph{enum#1}}%
  \else\if#2i%
    \@namedef{theenum#1}{\roman{enum#1}}%
  \else\if#2I%
    \@namedef{theenum#1}{\Roman{enum#1}}%
  \else\if#2o%
    \@namedef{theenum#1}{\arabic{enum#1}\protect\es@sptext{o}}%
  \fi\fi\fi\fi\fi\fi
  \toks@\expandafter{\csname theenum#1\endcsname}
  \expandafter\edef\csname labelenum#1\endcsname
     {\noexpand\es@listquot\the\toks@#3}}

\addto\layoutspanish{%
  \def\es@enumerate##1##2##3##4{%
    \es@enumdef{i}##1\@empty\@empty\@@
    \es@enumdef{ii}##2\@empty\@empty\@@
    \es@enumdef{iii}##3\@empty\@empty\@@
    \es@enumdef{iv}##4\@empty\@empty\@@}%
  \def\es@itemize##1##2##3##4{%
    \def\labelitemi{\es@listquot##1}%
    \def\labelitemii{\es@listquot##2}%
    \def\labelitemiii{\es@listquot##3}%
    \def\labelitemiv{\es@listquot##4}}%
  \def\p@enumii{\theenumi}%
  \def\p@enumiii{\theenumi\theenumii}%
  \def\p@enumiv{\p@enumiii\theenumiii}%
  \expandafter\es@enumerate\es@enum
  \expandafter\es@itemize\es@item
  \DeclareTextCommand{\guillemotleft}{OT1}{%
    \ifmmode\ll
    \else
```

```
      \save@sf@q{\penalty\@M
        \leavevmode\hbox{\usefont{U}{lasy}{m}{n}%
          \char40 \kern-0.19em\char40 }}%
    \fi}%
  \DeclareTextCommand{\guillemotright}{OT1}{%
    \ifmmode\gg
    \else
      \save@sf@q{\penalty\@M
          \leavevmode\hbox{\usefont{U}{lasy}{m}{n}%
            \char41 \kern-0.19em\char41 }}%
    \fi}%
  \def\@fnsymbol##1%
    {\ifcase##1\or*\or**\or***\or****\or
     *****\or******\else\@ctrerr\fi}%
  \def\@alph##1%
    {\ifcase##1\or a\or b\or c\or d\or e\or f\or g\or h\or i\or j\or
     k\or l\or m\or n\or \~n\or o\or p\or q\or r\or s\or t\or u\or v\or
     w\or x\or y\or z\else\@ctrerr\fi}%
  \def\@Alph##1%
    {\ifcase##1\or A\or B\or C\or D\or E\or F\or G\or H\or I\or J\or
     K\or L\or M\or N\or \~N\or O\or P\or Q\or R\or S\or T\or U\or V\or
     W\or X\or Y\or Z\else\@ctrerr\fi}%
  \let\@afterindentfalse\@afterindenttrue
  \@afterindenttrue
  \def\@seccntformat##1{\csname the##1\endcsname.\quad}%
  \def\numberline##1{\hb@xt@\@tempdima{##1\if&##1&\else.\fi\hfil}}%
  \def\@roman##1{\protect\es@roman{\number##1}}%
  \def\es@roman##1{\protect\es@lsc{\romannumeral##1}}%
  \def\esromanindex##1##2{##1{\protect\es@lsc{##2}}}}
```

We need to execute the following code when babel has been run, in order to see if `spanish` is the main language.

```
\AtEndOfPackage{%
  \let\es@activeacute\@undefined
  \def\bbl@tempa{spanish}%
  \ifx\bbl@main@language\bbl@tempa
    \AtBeginDocument{\layoutspanish}%
    \addto\es@select{%
      \@ifstar{\let\layoutspanish\relax}%
              {\layoutspanish\let\layoutspanish\relax}}%
  \fi
  \selectspanish}

\fi            %<<<<<<
```

After restoring the catcode of ~ and setting the minimal values for hyphenation, the `.ldf` is finished.

```
\es@savedcatcodes

\providehyphenmins{\CurrentOption}{\tw@\tw@}

```

```
\ifes@LaTeXe   %<<<<<<
  \ldf@finish{spanish}
\else          %<<<<<<
  \es@select
  \ldf@finish{spanish}
  \csname activatequoting\endcsname
\fi            %<<<<<<

⟨/code⟩
```

That's all in the main file. Now the file with custom-bib macros.

```
⟨*bblbst⟩
\def\bbland{y}
\def\bbleditors{directores}     \def\bbleds{dirs.\@}
\def\bbleditor{director}        \def\bbled{dir.\@}
\def\bbledby{dirigido por}
\def\bbledition{edici\'on}      \def\bbledn{ed.\@}
\def\bbletal{y otros}
\def\bblvolume{vol\'umen}       \def\bblvol{vol.\@}
\def\bblof{de}
\def\bblnumber{n\'umero}        \def\bblno{n\sptext{o}}
\def\bblin{en}
\def\bblpages{p\'aginas}        \def\bblpp{p\'ags.\@}
\def\bblpage{p\'gina}           \def\bblp{p\'ag.\@}
\def\bblchapter{cap\'itulo}     \def\bblchap{cap.\@}
\def\bbltechreport{informe t\'ecnico}
\def\bbltechrep{inf.\@ t\'ec.\@}
\def\bblmthesis{proyecto de fin de carrera}
\def\bblphdthesis{tesis doctoral}
\def\bblfirst {primera}         \def\bblfirsto {1\sptext{a}}
\def\bblsecond{segunda}         \def\bblsecondo{2\sptext{a}}
\def\bblthird {tercera}         \def\bblthirdo {3\sptext{a}}
\def\bblfourth{cuarta}          \def\bblfourtho{4\sptext{a}}
\def\bblfifth {quinta}          \def\bblfiftho {5\sptext{a}}
\def\bblth{\sptext{a}}
\let\bblst\bblth   \let\bblnd\bblth   \let\bblrd\bblth
\def\bbljan{enero}  \def\bblfeb{febrero}  \def\bblmar{marzo}
\def\bblapr{abril}  \def\bblmay{mayo}     \def\bbljun{junio}
\def\bbljul{julio}  \def\bblaug{agosto}   \def\bblsep{septiembre}
\def\bbloct{octubre}\def\bblnov{noviembre}\def\bbldec{diciembre}
⟨/bblbst⟩
```

The `spanish` option writes a macro in the page field of *MakeIndex* in entries with small caps number, and they are rejected. This program is a preprocessor which moves this macro to the entry field.

```
⟨*indexes⟩
\makeatletter

\newcount\es@converted
\newcount\es@processed

```

```
\def\es@encap{`\|}
\def\es@openrange{`\(}
\def\es@closerange{`\)}

\def\es@split@file#1.#2\@@{#1}
\def\es@split@ext#1.#2\@@{#2}

\typein[\answer]{^^JArchivo que convertir^^J%
   (extension por omision .idx):}

\@expandtwoargs\in@{.}{\answer}
\ifin@
  \edef\es@input@file{\expandafter\es@split@file\answer\@@}
  \edef\es@input@ext{\expandafter\es@split@ext\answer\@@}
\else
  \edef\es@input@file{\answer}
  \def\es@input@ext{idx}
\fi

\typein[\answer]{^^JArchivo de destino^^J%
   (archivo por omision: \es@input@file.eix,^^J%
    extension por omision .eix):}
\ifx\answer\@empty
  \edef\es@output{\es@input@file.eix}
\else
  \@expandtwoargs\in@{.}{\answer}
  \ifin@
     \edef\es@output{\answer}
  \else
     \edef\es@output{\answer.eix}
  \fi
\fi

\typein[\answer]{%
 ^^J?Se ha usado algun esquema especial de controles^^J%
 de MakeIndex para encap, open_range o close_range?^^J%
 [s/n] (n por omision)}

\if s\answer
  \typein[\answer]{^^JCaracter para 'encap'^^J%
    (\string| por omision)}
  \ifx\answer\@empty\else
    \edef\es@encap{%
      `\expandafter\noexpand\csname\expandafter\string\answer\endcsname}
  \fi
  \typein[\answer]{^^JCaracter para 'open_range'^^J%
    (\string( por omision)}
  \ifx\answer\@empty\else
    \edef\es@openrange{%
      `\expandafter\noexpand\csname\expandafter\string\answer\endcsname}
  \fi
  \typein[\answer]{^^JCaracter para 'close_range'^^J%
    (\string) por omision)}
  \ifx\answer\@empty\else
    \edef\es@closerange{%
      `\expandafter\noexpand\csname\expandafter\string\answer\endcsname}
  \fi
\fi

\newwrite\es@indexfile
\immediate\openout\es@indexfile=\es@output

\newif\ifes@encapsulated

\def\es@roman#1{\romannumeral#1 }
\edef\es@slash{\expandafter\@gobble\string\\}

\def\indexentry{%
  \begingroup
  \@sanitize
  \es@indexentry}

\begingroup

\catcode`\|=12 \lccode`\|=\es@encap\relax
\catcode`\(=12 \lccode`\(=\es@openrange\relax
\catcode`\)=12 \lccode`\)=\es@closerange\relax

\lowercase{
\gdef\es@indexentry#1{%
  \endgroup
  \advance\es@processed\@ne
  \es@encapsulatedfalse
  \es@bar@idx#1|\@@
  \es@idxentry}%
}

\lowercase{
\gdef\es@idxentry#1{%
  \in@{\es@roman}{#1}%
  \ifin@
    \advance\es@converted\@ne
    \immediate\write\es@indexfile{%
      \string\indexentry{\es@b|\ifes@encapsulated\es@p\fi esromanindex%
        {\ifx\es@a\@empty\else\es@slash\es@a\fi}}{#1}}%
  \else
    \immediate\write\es@indexfile{%
      \string\indexentry{\es@b\ifes@encapsulated|\es@p\es@a\fi}{#1}}%
  \fi}
}
```

```

\lowercase{
\gdef\es@bar@idx#1|#2\@@{%
  \def\es@b{#1}\def\es@a{#2}%
  \ifx\es@a\@empty\else\es@encapsulatedtrue\es@bar@eat#2\fi}
}

\lowercase{
\gdef\es@bar@eat#1#2|{\def\es@p{#1}\def\es@a{#2}%
  \edef\es@t{(}\ifx\es@t\es@p
  \else\edef\es@t{)}\ifx\es@t\es@p
  \else
    \edef\es@a{\es@p\es@a}\let\es@p\@empty%
  \fi\fi}
}

\endgroup

\input \es@input@file.\es@input@ext

\immediate\closeout\es@indexfile

\typeout{*****************}
\typeout{Se ha procesado: \es@input@file.\es@input@ext }
\typeout{Lineas leidas: \the\es@processed}
\typeout{Lineas convertidas: \the\es@converted}
\typeout{Resultado en: \es@output}
\ifnum\es@converted>\z@
  \typeout{Genere el indice a partir de ese archivo}
\else
  \typeout{No se ha realizado ningun tipo de conversion}
  \typeout{Se puede generar el archivo directamente^^J%
           de \es@input@file.\es@input@ext}
\fi
\typeout{*****************}
\@@end
⟨/indexes⟩
```

## 36 The Catalan language

The file `catalan.dtx`[32] defines all the language-specific macro's for the Catalan language.

For this language only the double quote character (`"`) is made active by default. In table 10 an overview is given of the new macros defined and the new meanings of `"`. Additionally to that, the user can explicitly activate the acute accent or apostrophe (`'`) and/or the grave accent (`` ` ``) characters by using the activeacute and activegrave options. In that case, the definitions shown in table 11 also become available[33].

| | |
|---|---|
| `\l.l` | geminated-l digraph (similar to l·l). `\L.L` produces the uppercase version. |
| `\lgem` | geminated-l digraph (similar to l·l). `\Lgem` produces the uppercase version. |
| `\up` | Macro to help typing raised ordinals, like 1$^{\text{er}}$. Takes one argument. |
| `\-` | like the old `\-`, but allowing hyphenation in the rest of the word. |
| `"i` | i with diaeresis, allowing hyphenation in the rest of the word. Valid for the following vowels: i, u (both lowercase and uppercase). |
| `"c` | c-cedilla (ç). Valid for both uppercase and lowercase c. |
| `"l` | geminated-l digraph (similar to l·l). Valid for both uppercase and lowercase l. |
| `"<` | French left double quotes (similar to <<). |
| `">` | French right double quotes (similar to >>). |
| `"-` | explicit hyphen sign, allowing hyphenation in the rest of the word. |
| `"\|` | disable ligature at this position. |

Table 10: Extra definitions made by file `catalan.ldf` (activated by default)

These active accents characters behave according to their original definitions if not followed by one of the characters indicated in that table.

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
36.1 ⟨*code⟩
36.2 \LdfInit{catalan}\captionscatalan
```

When this file is read as an option, i.e. by the `\usepackage` command, `catalan` could be an 'unknown' language in which case we have to make it known. So we

[32]The file described in this section has version number v2.2p and was last revised on 2005/03/29.

[33]Please note that if the acute accent character is active, it is necessary to take special care of coding apostrophes in a way which cannot be confounded with accents. Therefore, it is necessary to type `l'{}estri` instead of `l'estri`.

| | |
|---|---|
| `'e` | acute accented a, allowing hyphenation in the rest of the word. Valid for the following vowels: e, i, o, u (both lowercase and uppercase). |
| `` `a `` | grave accented a, allowing hyphenation in the rest of the word. Valid for the following vowels: a, e, o (both lowercase and uppercase). |

Table 11: Extra definitions made by file `catalan.ldf` (activated only when using the options activeacute and activegrave)

check for the existence of `\l@catalan` to see whether we have to do something here.

```
\ifx\l@catalan\@undefined
  \@nopatterns{Catalan}
  \adddialect\l@catalan0
\fi
```

The next step consists of defining commands to switch to (and from) the Catalan language.

`\catalanhyphenmins` This macro is used to store the correct values of the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`.

```
\providehyphenmins{catalan}{\tw@\tw@}
```

`\captionscatalan` The macro `\captionscatalan` defines all strings used in the four standard documentclasses provided with LaTeX.

```
\addto\captionscatalan{%
  \def\prefacename{Pr\`oleg}%
  \def\refname{Refer\`encies}%
  \def\abstractname{Resum}%
  \def\bibname{Bibliografia}%
  \def\chaptername{Cap\'{\i}tol}%
  \def\appendixname{Ap\`endix}%
  \def\contentsname{\'Index}%
  \def\listfigurename{\'Index de figures}%
  \def\listtablename{\'Index de taules}%
  \def\indexname{\'Index alfab\`etic}%
  \def\figurename{Figura}%
  \def\tablename{Taula}%
  \def\partname{Part}%
  \def\enclname{Adjunt}%
  \def\ccname{C\`opies a}%
  \def\headtoname{A}%
  \def\pagename{P\`agina}%
  \def\seename{Vegeu}%
  \def\alsoname{Vegeu tamb\'e}%
  \def\proofname{Demostraci\'o}%
  \def\glossaryname{Glossari}%
}
```

`\datecatalan` The macro `\datecatalan` redefines the command `\today` to produce Catalan dates. Months are written in lowercase[34].

```
\def\datecatalan{%
  \def\today{\number\day~\ifcase\month\or
    de gener\or de febrer\or de mar\c{c}\or d'abril\or de maig\or
    de juny\or de juliol\or d'agost\or de setembre\or d'octubre\or
    de novembre\or de desembre\fi
    \space de~\number\year}}
```

`\extrascatalan` `\noextrascatalan` The macro `\extrascatalan` will perform all the extra definitions needed for the Catalan language. The macro `\noextrascatalan` is used to cancel the actions of `\extrascatalan`.

To improve hyphenation we give the grave character (`'`) a non-zero lower case code; when we do that TeX will find more breakpoints in words that contain this character in its rôle as apostrophe.

```
\addto\extrascatalan{%
  \lccode`'=`'}
\addto\noextrascatalan{%
  \lccode`'=0}
```

For Catalan, some characters are made active or are redefined. In particular, the `"` character receives a new meaning; this can also happen for the `'` character and the `` ` `` character when the options activegrave and/or activeacute are specified.

```
\addto\extrascatalan{\languageshorthands{catalan}}
\initiate@active@char{"}
\addto\extrascatalan{\bbl@activate{"}}
```

Because the grave character is being used in constructs such as ``\catcode``=\active`` it needs to have it's original category code" when the auxiliary file is being read. Note that this file is read twice, once at the beginning of the document; then there is no problem; but the second time it is read at the end of the document to check whether any labels changes. It's this second time round that the actived grave character leads to error messages.

```
\@ifpackagewith{babel}{activegrave}{%
  \AtBeginDocument{%
    \if@filesw\immediate\write\@auxout{\catcode096=12}\fi}
  \initiate@active@char{`}%
  }{}
\@ifpackagewith{babel}{activegrave}{%
  \addto\extrascatalan{\bbl@activate{`}}%
  }{}
\@ifpackagewith{babel}{activeacute}{%
  \initiate@active@char{'}%
  }{}
```

[34]This seems to be the common practice. See for example: E. Coromina, *El 9 Nou: Manual de redacció i estil*, Ed. Eumo, Vic, 1993

```
\@ifpackagewith{babel}{activeacute}{%
  \addto\extrascatalan{\bbl@activate{'}}%
  }{}
```

Now make sure that the characters that have been turned into shorthanfd characters expand to 'normal' characters outside the catalan environment.

```
\addto\noextrascatalan{\bbl@deactivate{"}}
\@ifpackagewith{babel}{activegrave}{%
  \addto\noextrascatalan{\bbl@deactivate{`}}}{}
\@ifpackagewith{babel}{activeacute}{%
  \addto\noextrascatalan{\bbl@deactivate{'}}}{}
```

Apart from the active characters some other macros get a new definition. Therefore we store the current ones to be able to restore them later. When their current meanings are saved, we can safely redefine them.

We provide new definitions for the accent macros when one or both of the options activegrave or activeacute were specified.

```
\addto\extrascatalan{%
  \babel@save\"%
  \def\"{\protect\@umlaut}}%
\@ifpackagewith{babel}{activegrave}{%
  \babel@save\`%
  \addto\extrascatalan{\def\`{\protect\@grave}}
  }{}
\@ifpackagewith{babel}{activeacute}{%
  \babel@save\'%
  \addto\extrascatalan{\def\'{\protect\@acute}}
  }{}
```

All the code above is necessary because we need a few extra active characters. These characters are then used as indicated in tables 10 and 11.

`\dieresis` `\textacute` `\textgrave` The original definition of `\"` is stored as `\dieresis`, because the definition of `\"` might not be the default plain TEX one. If the user uses PostScript fonts with the Adobe font encoding the `"` character is not in the same position as in Knuth's font encoding. In this case `\"` will not be defined as `\accent"7F 1`, but as `\accent'310 #1`. Something similar happens when using fonts that follow the Cork encoding. For this reason we save the definition of `\"` and use that in the definition of other macros. We do likewise for `` \` ``, and `\'`.

```
\let\dieresis\"
\@ifpackagewith{babel}{activegrave}{\let\textgrave\`}{}
\@ifpackagewith{babel}{activeacute}{\let\textacute\'}{}
```

`\@umlaut` `\@acute` `\@grave` We check the encoding and if not using T1, we make the accents expand but enabling hyphenation beyond the accent. If this is the case, not all break positions will be found in words that contain accents, but this is a limitation in TEX. An unsolved problem here is that the encoding can change at any time. The definitions below are made in such a way that a change between two 256-char encodings are supported, but changes between a 128-char and a 256-char encoding are not properly supported. We check if T1 is in use. If not, we will give a warning and proceed redefining the accent macros so that TEX at least finds the breaks that are not too close to the accent. The warning will only be printed to the log file.

```
\ifx\DeclareFontShape\@undefined
  \wlog{Warning: You are using an old LaTeX}
  \wlog{Some word breaks will not be found.}
  \def\@umlaut#1{\allowhyphens\dieresis{#1}\allowhyphens}
  \@ifpackagewith{babel}{activeacute}{%
    \def\@acute#1{\allowhyphens\textacute{#1}\allowhyphens}}{}
  \@ifpackagewith{babel}{activegrave}{%
    \def\@grave#1{\allowhyphens\textgrave{#1}\allowhyphens}}{}
\else
  \ifx\f@encoding\bbl@t@one
    \let\@umlaut\dieresis
    \@ifpackagewith{babel}{activeacute}{%
      \let\@acute\textacute}{}
    \@ifpackagewith{babel}{activegrave}{%
      \let\@grave\textgrave}{}
  \else
    \wlog{Warning: You are using encoding \f@encoding\space
      instead of T1.}
    \wlog{Some word breaks will not be found.}
    \def\@umlaut#1{\allowhyphens\dieresis{#1}\allowhyphens}
    \@ifpackagewith{babel}{activeacute}{%
      \def\@acute#1{\allowhyphens\textacute{#1}\allowhyphens}}{}
    \@ifpackagewith{babel}{activegrave}{%
      \def\@grave#1{\allowhyphens\textgrave{#1}\allowhyphens}}{}
  \fi
\fi
```

If the user setup has extended fonts, the Ferguson macros are required to be defined. We check for their existance and, if defined, expand to whatever they are defined to. For instance, `\'a` would check for the existance of a `\@ac@a` macro. It is assumed to expand to the code of the accented letter. If it is not defined, we assume that no extended codes are available and expand to the original definition but enabling hyphenation beyond the accent. This is as best as we can do. It is better if you have extended fonts or ML-TEX because the hyphenation algorithm can work on the whole word. The following macros are directly derived from ML-TEX.[35]

Now we can define our shorthands: the diaeresis and "ela geminada" support,

```
\declare@shorthand{catalan}{"i}{\textormath{\@umlaut\i}{\ddot\imath}}
```

[35]A problem is perceived here with these macros when used in a multilingual environment where extended hyphenation patterns are available for some but not all languages. Assume that no extended patterns exist at some site for French and that `french.sty` would adopt this scheme too. In that case, `'e` in French would produce the combined accented letter, but hyphenation around it would be suppressed. Both language options would need an independent method to know whether they have extended patterns available. The precise impact of this problem and the possible solutions are under study.

```
\declare@shorthand{catalan}{"l}{\lgem{}}
\declare@shorthand{catalan}{"u}{\textormath{\@umlaut u}{\ddot u}}
\declare@shorthand{catalan}{"I}{\textormath{\@umlaut I}{\ddot I}}
\declare@shorthand{catalan}{"L}{\Lgem{}}
\declare@shorthand{catalan}{"U}{\textormath{\@umlaut U}{\ddot U}}
```

cedille,

```
\declare@shorthand{catalan}{"c}{\textormath{\c c}{^{\prime} c}}
\declare@shorthand{catalan}{"C}{\textormath{\c C}{^{\prime} C}}
```

'french' quote characters,

```
\declare@shorthand{catalan}{"<}{%
  \textormath{\guillemotleft}{\mbox{\guillemotleft}}}
\declare@shorthand{catalan}{">}{%
  \textormath{\guillemotright}{\mbox{\guillemotright}}}
```

grave accents,

```
\@ifpackagewith{babel}{activegrave}{%
  \declare@shorthand{catalan}{`a}{\textormath{\@grave a}{\grave a}}
  \declare@shorthand{catalan}{`e}{\textormath{\@grave e}{\grave e}}
  \declare@shorthand{catalan}{`o}{\textormath{\@grave o}{\grave o}}
  \declare@shorthand{catalan}{`A}{\textormath{\@grave A}{\grave A}}
  \declare@shorthand{catalan}{`E}{\textormath{\@grave E}{\grave E}}
  \declare@shorthand{catalan}{`O}{\textormath{\@grave O}{\grave O}}
  \declare@shorthand{catalan}{``}{\textquotedblleft}%''
  }{}
```

acute accents,

```
\@ifpackagewith{babel}{activeacute}{%
  \declare@shorthand{catalan}{'a}{\textormath{\@acute a}{^{\prime} a}}
  \declare@shorthand{catalan}{'e}{\textormath{\@acute e}{^{\prime} e}}
  \declare@shorthand{catalan}{'i}{\textormath{\@acute\i{}}{^{\prime} i}}
  \declare@shorthand{catalan}{'o}{\textormath{\@acute o}{^{\prime} o}}
  \declare@shorthand{catalan}{'u}{\textormath{\@acute u}{^{\prime} u}}
  \declare@shorthand{catalan}{'A}{\textormath{\@acute A}{^{\prime} A}}
  \declare@shorthand{catalan}{'E}{\textormath{\@acute E}{^{\prime} E}}
  \declare@shorthand{catalan}{'I}{\textormath{\@acute I}{^{\prime} I}}
  \declare@shorthand{catalan}{'O}{\textormath{\@acute O}{^{\prime} O}}
  \declare@shorthand{catalan}{'U}{\textormath{\@acute U}{^{\prime} U}}
  \declare@shorthand{catalan}{'|}{%
    \textormath{\csname normal@char\string'\endcsname}{^{\prime}}}
```

the acute accent,

```
  \declare@shorthand{catalan}{''}{%
    \textormath{\textquotedblright}{\sp\bgroup\prim@s'}}
  }{}
```

and finally, some support definitions

```
\declare@shorthand{catalan}{"-}{\nobreak-\bbl@allowhyphens}
\declare@shorthand{catalan}{"|}{%
  \textormath{\nobreak\discretionary{-}{}{\kern.03em}%
              \allowhyphens}{}}
```

\- All that is left now is the redefinition of \-. The new version of \- should indicate an extra hyphenation position, while allowing other hyphenation positions to be generated automatically. The standard behaviour of TEX in this respect is unfortunate for Catalan but not as much as for Dutch or German, where long compound words are quite normal and all one needs is a means to indicate an extra hyphenation position on top of the ones that TEX can generate from the hyphenation patterns. However, the average length of words in Catalan makes this desirable and so it is kept here.

```
\addto\extrascatalan{%
  \babel@save{\-}%
  \def\-{\bbl@allowhyphens\discretionary{-}{}{}\bbl@allowhyphens}}
```

\lgem \Lgem Here we define a macro for typing the catalan "ela geminada" (geminated l). The macros \lgem and \Lgem have been chosen for its lowercase and uppercase representation, respectively[36].

The code used in the actual macro used is a combination of the one proposed by Feruglio and Fuster[37] and the proposal[38] from Valiente presented at the TEX Users Group Annual Meeting in 1995. This last proposal has not been fully implemented due to its limitation to CM fonts.

```
\newdimen\leftllkern \newdimen\rightllkern \newdimen\raiselldim
\def\lgem{%
  \ifmmode
    \csname normal@char\string"\endcsname l%
  \else
    \leftllkern=0pt\rightllkern=0pt\raiselldim=0pt%
    \setbox0\hbox{l}\setbox1\hbox{l\/}\setbox2\hbox{.}%
    \advance\raiselldim by \the\fontdimen5\the\font
    \advance\raiselldim by -\ht2%
    \leftllkern=-.25\wd0%
    \advance\leftllkern by \wd1%
    \advance\leftllkern by -\wd0%
    \rightllkern=-.25\wd0%
    \advance\rightllkern by -\wd1%
    \advance\rightllkern by \wd0%
    \allowhyphens\discretionary{l-}{l}%
    {\hbox{l}\kern\leftllkern\raise\raiselldim\hbox{.}%
      \kern\rightllkern\hbox{l}}\allowhyphens
  \fi
  }
\def\Lgem{%
  \ifmmode
    \csname normal@char\string"\endcsname L%
  \else
    \leftllkern=0pt\rightllkern=0pt\raiselldim=0pt%
```

[36]The macro names \ll and \LL were not taken because of the fact that \ll is already used in mathematical mode.

[37]G. Valiente and R. Fuster, Typesetting Catalan Texts with TEX, *TUGboat* **14**(3), 1993.

[38]G. Valiente, Modern Catalan Typographical Conventions, *TUGboat* **16**(3), 1995.

```
    \setbox0\hbox{L}\setbox1\hbox{L\/}\setbox2\hbox{.}%
    \advance\raiselldim by .5\ht0%
    \advance\raiselldim by -.5\ht2%
    \leftllkern=-.125\wd0%
    \advance\leftllkern by \wd1%
    \advance\leftllkern by -\wd0%
    \rightllkern=-\wd0%
    \divide\rightllkern by 6%
    \advance\rightllkern by -\wd1%
    \advance\rightllkern by \wd0%
    \allowhyphens\discretionary{L-}{L}%
    {\hbox{L}\kern\leftllkern\raise\raiselldim\hbox{.}%
      \kern\rightllkern\hbox{L}}\allowhyphens
  \fi
  }
```

`\l.l` `\L.L` It seems to be the most natural way of entering the "ela geminda" to use the sequences `\l.l` and `\L.L`. These are not really macro's by themselves but the macros `\l` and `\L` with delimited arguments. Therefor we define two macros that check if the next character is a period. If not the "polish l" will be typeset, otherwise a "ela geminada" will be typeset and the next two tokens will be 'eaten'.

```
\AtBeginDocument{%
  \let\lslash\l
  \let\Lslash\L
  \DeclareRobustCommand\l{\@ifnextchar.\bbl@l\lslash}
  \DeclareRobustCommand\L{\@ifnextchar.\bbl@L\Lslash}}
\def\bbl@l#1#2{\lgem}
\def\bbl@L#1#2{\Lgem}
```

`\up` A macro for typesetting things like 1$^{\text{er}}$ as proposed by Raymon Seroul[39].

```
\DeclareRobustCommand*{\up}[1]{\textsuperscript{#1}}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{catalan}
⟨/code⟩
```

[39]This macro has been borrowed from francais.dtx

## 37 The Galician language

The file `galician.dtx`[40] defines all the language definition macros for the Galician language.

For this language the characters `'` `~` and `"` are made active. In table 12 an overview is given of their purpose. These active accents character behave according

| | |
|---|---|
| `"\|` | disable ligature at this position. |
| `"-` | an explicit hyphen sign, allowing hyphenation in the rest of the word. |
| `\-` | like the old `\-`, but allowing hyphenation in the rest of the word. |
| `'a` | an accent that allows hyphenation. Valid for all vowels uppercase and lowercase. |
| `'n` | a n with a tilde. This is included to improve compatibility with FTC. Works for uppercase too. |
| `"u` | a u with dieresis allowing hyphenation. |
| `"a` | feminine ordinal as in 1$^{\underline{a}}$. |
| `"o` | masculine ordinal as in 1$^{\underline{o}}$. |
| `~n` | a n with tilde. Works for uppercase too. |

Table 12: The extra definitions made by `galician.ldf`

to their original definitions if not followed by one of the characters indicated in that table.

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
37.1 ⟨*code⟩
37.2 \LdfInit{galician}\captionsgalician
```

When this file is read as an option, i.e. by the `\usepackage` command, `galician` could be an 'unknown' language in which case we have to make it known. So we check for the existence of `\l@galician` to see whether we have to do something here.

```
37.3 \ifx\l@galician\@undefined
37.4   \@nopatterns{Galician}
37.5   \adddialect\l@galician0\fi
```

The next step consists of defining commands to switch to (and from) the Galician language.

`\captionsgalician` The macro `\captionsgalician` defines all strings used in the four standard documentclasses provided with LaTeX.

```
37.6 \addto\captionsgalician{%
37.7   \def\prefacename{Prefacio}%
37.8   \def\refname{Referencias}%
```

[40] The file described in this section has version number v1.2l and was last revised on 2005/03/30.

```
  \def\abstractname{Resumo}%
  \def\bibname{Bibliograf\'{\i}a}%
  \def\chaptername{Cap\'{\i}tulo}%
  \def\appendixname{Ap\'endice}%
  \def\contentsname{\'Indice Xeral}%
  \def\listfigurename{\'Indice de Figuras}%
  \def\listtablename{\'Indice de T\'aboas}%
  \def\indexname{\'Indice de Materias}%
  \def\figurename{Figura}%
  \def\tablename{T\'aboa}%
  \def\partname{Parte}%
  \def\enclname{Adxunto}%
  \def\ccname{Copia a}%
  \def\headtoname{A}%
  \def\pagename{P\'axina}%
  \def\seename{v\'exase}%
  \def\alsoname{v\'exase tam\'en}%
  \def\proofname{Demostraci\'on}%
  \def\glossaryname{Glosario}%
}
```

`\dategalician` The macro `\dategalician` redefines the command `\today` to produce Galician dates.

```
\def\dategalician{%
  \def\today{\number\day~de\space\ifcase\month\or
    xaneiro\or febreiro\or marzo\or abril\or maio\or xu\~no\or
    xullo\or agosto\or setembro\or outubro\or novembro\or decembro\fi
    \space de~\number\year}}
```

`\extrasgalician` `\noextrasgalician` The macro `\extrasgalician` will perform all the extra definitions needed for the Galician language. The macro `\noextrasgalician` is used to cancel the actions of `\extrasgalician`.

For Galician, some characters are made active or are redefined. In particular, the `"` character and the `~` character receive new meanings this can also happen for the `'` character when the option activeacute is specified.

```
\addto\extrasgalician{\languageshorthands{galician}}
\initiate@active@char{"}
\initiate@active@char{~}
\addto\extrasgalician{%
  \bbl@activate{"}\bbl@activate{~}}
\@ifpackagewith{babel}{activeacute}{%
  \initiate@active@char{'}}{}
\@ifpackagewith{babel}{activeacute}{%
  \addto\extrasgalician{\bbl@activate{'}}}{}
```

Now make sure that the characters that have been turned into shorthanfd characters expand to 'normal' characters outside the catalan environment.

```
\addto\noextrasgalician{%
  \bbl@deactivate{"}\bbl@deactivate{~}}
```

```
\@ifpackagewith{babel}{activeacute}{%
  \addto\noextrascatalan{\bbl@deactivate{'}}}{}
```

Apart from the active characters some other macros get a new definition. Therefore we store the current one to be able to restore them later.

```
\addto\extrasgalician{%
  \babel@save\"\babel@save\~%
  \def\"{\protect\@umlaut}%
  \def\~{\protect\@tilde}}
\@ifpackagewith{babel}{activeacute}{%
  \babel@save\'%
  \addto\extrasgalician{\def\'{\protect\@acute}}
  }{}
```

All the code above is necessary because we need a few extra active characters. These characters are then used as indicated in table 12.

This option includes some support for working with extended, 8-bit fonts, if available. This assumes that the user has some macros predefined. For instance, if the user has a `\@ac@a` macro defined, the sequence `\'a` or `'a` will both expand to whatever `\@ac@a` is defined to expand, presumably á. The names of these macros are the same as those in Ferguson's ML-TEX compatibility package on purpose. Using this method, and provided that adequate hyphenation patterns exist, it is possible to get better hyphenation for Galician than before. If the user has a terminal able to produce these codes directly, it is possible to do so. If the need arises to send the document to someone who does not have such support, it is possible to mechanically translate the document so that the receiver can make use of it.

To be able to define the function of the new accents, we first define a couple of 'support' macros.

`\dieresis` `\textacute` `\texttilde`

The original definition of `\"` is stored as `\dieresis`, because the definition of `\"` might not be the default plain TEX one. If the user uses PostScript fonts with the Adobe font encoding the `"` character is not in the same position as in Knuth's font encoding. In this case `\"` will not be defined as `\accent"7F #1`, but as `\accent'310 #1`. Something similar happens when using fonts that follow the Cork encoding. For this reason we save the definition of `\"` and use that in the definition of other macros. We do likewise for `\'` and `\~`.

```
\let\dieresis\"
\let\texttilde\~
\@ifpackagewith{babel}{activeacute}{\let\textacute\'}{}
```

`\@umlaut` `\@acute` `\@tilde`

If the user setup has extended fonts, the Ferguson macros are required to be defined. We check for their existance and, if defined, expand to whatever they are defined to. For instance, `\'a` would check for the existance of a `\@ac@a` macro. It is assumed to expand to the code of the accented letter. If it is not defined, we assume that no extended codes are available and expand to the original definition but enabling hyphenation beyond the accent. This is as best as we can do. It is better if you have extended fonts or ML-TEX because the hyphenation algorithm can work on the whole word. The following macros are directly derived from ML-TEX.[41]

```
\def\@umlaut#1{\allowhyphens\dieresis{#1}\allowhyphens}
\def\@tilde#1{\allowhyphens\texttilde{#1}\allowhyphens}
\@ifpackagewith{babel}{activeacute}{%
  \def\@acute#1{\allowhyphens\textacute{#1}\allowhyphens}}{}
```

Now we can define our shorthands: the umlauts,

```
\declare@shorthand{galician}{"-}{\nobreak-\bbl@allowhyphens}
\declare@shorthand{galician}{"|}{\discretionary{-}{}{\kern.03em}}
\declare@shorthand{galician}{"u}{\@umlaut{u}}
\declare@shorthand{galician}{"U}{\@umlaut{U}}
```

ordinals[42],

```
\declare@shorthand{galician}{"o}{%
  \leavevmode\raise1ex\hbox{\underbar{\scriptsize o}}}
\declare@shorthand{galician}{"a}{%
  \leavevmode\raise1ex\hbox{\underbar{\scriptsize a}}}
```

acute accents,

```
\@ifpackagewith{babel}{activeacute}{%
  \declare@shorthand{galician}{'a}{\textormath{\@acute a}{^{\prime} a}}
  \declare@shorthand{galician}{'e}{\textormath{\@acute e}{^{\prime} e}}
  \declare@shorthand{galician}{'i}{\textormath{\@acute\i{}}{^{\prime}i}}
  \declare@shorthand{galician}{'o}{\textormath{\@acute o}{^{\prime} o}}
  \declare@shorthand{galician}{'u}{\textormath{\@acute u}{^{\prime} u}}
  \declare@shorthand{galician}{'A}{\textormath{\@acute A}{^{\prime} A}}
  \declare@shorthand{galician}{'E}{\textormath{\@acute E}{^{\prime} E}}
  \declare@shorthand{galician}{'I}{\textormath{\@acute I}{^{\prime} I}}
  \declare@shorthand{galician}{'O}{\textormath{\@acute O}{^{\prime} O}}
  \declare@shorthand{galician}{'U}{\textormath{\@acute U}{^{\prime} U}}
```

tildes,

```
  \declare@shorthand{galician}{'n}{\textormath{\~n}{^{\prime} n}}
  \declare@shorthand{galician}{'N}{\textormath{\~N}{^{\prime} N}}
```

the acute accent,

```
  \declare@shorthand{galician}{''}{%
    \textormath{\textquotedblright}{\sp\bgroup\prim@s'}}
  }{}
\declare@shorthand{galician}{~n}{\textormath{\~n}{\@tilde n}}
\declare@shorthand{galician}{~N}{\textormath{\~N}{\@tilde N}}
```

[41]A problem is perceived here with these macros when used in a multilingual environment where extended hyphenation patterns are available for some but not all languages. Assume that no extended patterns exist at some site for French and that `french.sty` would adopt this scheme too. In that case, `'e` in French would produce the combined accented letter, but hyphenation around it would be suppressed. Both language options would need an independent method to know whether they have extended patterns available. The precise impact of this problem and the possible solutions are under study.

[42]The code for the ordinals was taken from the answer provided by Raymond Chen (raymond@math.berkeley.edu) to a question by Joseph Gil (`yogi@cs.ubc.ca`) in `comp.text.tex`.

\- All that is left now is the redefinition of \-. The new version of \- should indicate an extra hyphenation position, while allowing other hyphenation positions to be generated automatically. The standard behaviour of TEX in this respect is unfortunate for Galician but not as much as for Dutch or German, where long compound words are quite normal and all one needs is a means to indicate an extra hyphenation position on top of the ones that TEX can generate from the hyphenation patterns. However, the average length of words in Galician makes this desirable and so it is kept here.

```
\addto\extrasgalician{%
  \babel@save{\-}%
  \def\-{\bbl@allowhyphens\discretionary{-}{}{}\bbl@allowhyphens}}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{galician}
⟨/code⟩
```

## 38 The Basque language

The file `basque.dtx`[43] defines all the language definition macro's for the Basque language.

For this language the characters `~` and `"` are made active. In table 13 an overview is given of their purpose. These active accent characters behave according

| | |
|---|---|
| `"\|` | disable ligature at this position. |
| `"-` | an explicit hyphen sign, allowing hyphenation in the rest of the word. |
| `\-` | like the old `\-`, but allowing hyphenation in the rest of the word. |
| `"<` | for French left double quotes (similar to <<). |
| `">` | for French right double quotes (similar to >>). |
| `~n` | a n with tilde. Works for uppercase too. |

Table 13: The extra definitions made by `basque.ldf`

to their original definitions if not followed by one of the characters indicated in that table.

This option includes support for working with extended, 8-bit fonts, if available. Support is based on providing an appropriate definition for the accent macros on entry to the Basque language. This is automatically done by LaTeX 2ε or NFSS2. If T1 encoding is chosen, and provided that adequate hyphenation patterns[44] are available. The easiest way to use the new encoding with LaTeX 2ε is to load the package `t1enc` with `\usepackage`. This must be done before loading babel.

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
38.1 ⟨*code⟩
38.2 \LdfInit{basque}\captionsbasque
```

When this file is read as an option, i.e. by the `\usepackage` command, `basque` could be an 'unknown' language in which case we have to make it known. So we check for the existence of `\l@basque` to see whether we have to do something here.

```
38.3 \ifx\l@basque\@undefined
38.4   \@nopatterns{Basque}
38.5   \adddialect\l@basque0
38.6 \fi
```

The next step consists of defining commands to switch to (and from) the Basque language.

[43]The file described in this section has version number v1.0f and was last revised on 2005/03/29. The original author is Juan M. Aguirregabiria, (`wtpagagj@lg.ehu.es`) and is based on the Spanish file by Julio Sánchez, (`jsanchez@gmv.es`).

[44]One source for such patterns is the archive at `tp.lc.ehu.es` that can be accessed by anonymous FTP or in `http://tp.lc.ehu.es/jma/basque.html`

\captionsbasque The macro \captionsbasque defines all strings used in the four standard documentclasses provided with LaTeX.

```
\addto\captionsbasque{%
  \def\prefacename{Hitzaurrea}%
  \def\refname{Erreferentziak}%
  \def\abstractname{Laburpena}%
  \def\bibname{Bibliografia}%
  \def\chaptername{Kapitulua}%
  \def\appendixname{Eranskina}%
  \def\contentsname{Gaien Aurkibidea}%
  \def\listfigurename{Irudien Zerrenda}%
  \def\listtablename{Taulen Zerrenda}%
  \def\indexname{Kontzeptuen Aurkibidea}%
  \def\figurename{Irudia}%
  \def\tablename{Taula}%
  \def\partname{Atala}%
  \def\enclname{Erantsia}%
  \def\ccname{Kopia}%
  \def\headtoname{Nori}%
  \def\pagename{Orria}%
  \def\seename{Ikusi}%
  \def\alsoname{Ikusi, halaber}%
  \def\proofname{Frogapena}%
  \def\glossaryname{Glosarioa}%
  }%
```

\datebasque The macro \datebasque redefines the command \today to produce Basque

```
\def\datebasque{%
  \def\today{\number\year.eko\space\ifcase\month\or
    urtarrilaren\or otsailaren\or martxoaren\or apirilaren\or
    maiatzaren\or ekainaren\or uztailaren\or abuztuaren\or
    irailaren\or urriaren\or azaroaren\or
    abenduaren\fi~\number\day}}
```

\extrasbasque \noextrasbasque The macro \extrasbasque will perform all the extra definitions needed for the Basque language. The macro \noextrasbasque is used to cancel the actions of \extrasbasque. For Basque, some characters are made active or are redefined. In particular, the " character and the ~ character receive new meanings. Therefore these characters have to be treated as 'special' characters.

```
\addto\extrasbasque{\languageshorthands{basque}}
\initiate@active@char{"}
\initiate@active@char{~}
\addto\extrasbasque{%
  \bbl@activate{"}%
  \bbl@activate{~}}
```

Don't forget to turn the shorthands off again.

```
\addto\noextrasbasque{
  \bbl@deactivate{"}\bbl@deactivate{~}}
```

Apart from the active characters some other macros get a new definition. Therefore we store the current one to be able to restore them later.

```
\addto\extrasbasque{%
  \babel@save\"%
  \babel@save\~%
  \def\"{\protect\@umlaut}%
  \def\~{\protect\@tilde}}
```

`\basquehyphenmins` Basque hyphenation uses `\lefthyphenmin` and `\righthyphenmin` both set to 2.

```
\providehyphenmins{\CurrentOption}{\tw@\tw@}
```

`\dieresia` `\texttilde` The original definition of `\"` is stored as `\dieresia`, because the we do not know what is its definition, since it depends on the encoding we are using or on special macros that the user might have loaded. The expansion of the macro might use the TEX `\accent` primitive using some particular accent that the font provides or might check if a combined accent exists in the font. These two cases happen with respectively OT1 and T1 encodings. For this reason we save the definition of `\"` and use that in the definition of other macros. We do likewise for `\'` and `\~`. The present coding of this option file is incorrect in that it can break when the encoding changes. We do not use `\tilde` as the macro name because it is already defined as `\mathaccent`.

```
\let\dieresia\"
\let\texttilde\~
```

`\@umlaut` `\@tilde` We check the encoding and if not using T1, we make the accents expand but enabling hyphenation beyond the accent. If this is the case, not all break positions will be found in words that contain accents, but this is a limitation in TEX. An unsolved problem here is that the encoding can change at any time. The definitions below are made in such a way that a change between two 256-char encodings are supported, but changes between a 128-char and a 256-char encoding are not properly supported. We check if T1 is in use. If not, we will give a warning and proceed redefining the accent macros so that TEX at least finds the breaks that are not too close to the accent. The warning will only be printed to the log file.

```
\ifx\DeclareFontShape\@undefined
  \wlog{Warning: You are using an old LaTeX}
  \wlog{Some word breaks will not be found.}
  \def\@umlaut#1{\allowhyphens\dieresia{#1}\allowhyphens}
  \def\@tilde#1{\allowhyphens\texttilde{#1}\allowhyphens}
\else
  \edef\bbl@next{T1}
  \ifx\f@encoding\bbl@next
    \let\@umlaut\dieresia
    \let\@tilde\texttilde
  \else
    \wlog{Warning: You are using encoding \f@encoding\space
      instead of T1.}
    \wlog{Some word breaks will not be found.}
    \def\@umlaut#1{\allowhyphens\dieresia{#1}\allowhyphens}
    \def\@tilde#1{\allowhyphens\texttilde{#1}\allowhyphens}
  \fi
\fi
```

Now we can define our shorthands: the french quotes,

```
\declare@shorthand{basque}{"<}{%
  \textormath{\guillemotleft}{\mbox{\guillemotleft}}}
\declare@shorthand{basque}{">}{%
  \textormath{\guillemotright}{\mbox{\guillemotright}}}
```

ordinals[45],

```
  \declare@shorthand{basque}{''}{%
    \textormath{\textquotedblright}{\sp\bgroup\prim@s'}}
```

tildes,

```
\declare@shorthand{basque}{~n}{\textormath{\~n}{\@tilde n}}
\declare@shorthand{basque}{~N}{\textormath{\~N}{\@tilde N}}
```

and some additional commands.

The shorthand `"-` should be used in places where a word contains an explictit hyphenation character. According to the Academy of the Basque language, when a word break occurs at an explicit hyphen it must appear *both* at the end of the first line *and* at the beginning of the second line.

```
\declare@shorthand{basque}{"-}{%
  \nobreak\discretionary{-}{-}{-}\bbl@allowhyphens}
\declare@shorthand{basque}{"|}{%
  \textormath{\nobreak\discretionary{-}{}{\kern.03em}%
              \allowhyphens}{}}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{basque}
⟨/code⟩
```

[45]The code for the ordinals was taken from the answer provided by Raymond Chen (`raymond@math.berkeley.edu`) to a question by Joseph Gil (`yogi@cs.ubc.ca`) in `comp.text.tex`.

## 39 The Romanian language

The file `romanian.dtx`[46] defines all the language-specific macros for the Romanian language.

For this language currently no special definitions are needed or available.

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
⟨*code⟩
\LdfInit{romanian}\captionsromanian
```

When this file is read as an option, i.e. by the `\usepackage` command, `romanian` will be an 'unknown' language in which case we have to make it known. So we check for the existence of `\l@romanian` to see whether we have to do something here.

```
\ifx\l@romanian\@undefined
    \@nopatterns{Romanian}
    \adddialect\l@romanian0\fi
```

The next step consists of defining commands to switch to (and from) the Romanian language.

`\captionsromanian` The macro `\captionsromanian` defines all strings used in the four standard documentclasses provided with LaTeX.

```
ls
\addto\captionsromanian{%
  \def\prefacename{Prefa\c{t}\u{a}}%
  \def\refname{Bibliografie}%
  \def\abstractname{Rezumat}%
  \def\bibname{Bibliografie}%
  \def\chaptername{Capitolul}%
  \def\appendixname{Anexa}%
  \def\contentsname{Cuprins}%
  \def\listfigurename{List\u{a} de figuri}%
  \def\listtablename{List\u{a} de tabele}%
  \def\indexname{Glosar}%
  \def\figurename{Figura}%    % sau Plan\c{s}a
  \def\tablename{Tabela}%
  \def\partname{Partea}%
  \def\enclname{Anex\u{a}}%   % sau Anexe
  \def\ccname{Copie}%
  \def\headtoname{Pentru}%
  \def\pagename{Pagina}%
  \def\seename{Vezi}%
  \def\alsoname{Vezi de asemenea}%
  \def\proofname{Demonstra\c{t}ie} %
  \def\glossaryname{Glosar}%
  }%
```

[46] The file described in this section has version number v1.2l and was last revised on 2005/03/31. A contribution was made by Umstatter Horst (`hhu@cernvm.cern.ch`).

`\dateromanian` The macro `\dateromanian` redefines the command `\today` to produce Romanian dates.

```
\def\dateromanian{%
  \def\today{\number\day~\ifcase\month\or
    ianuarie\or februarie\or martie\or aprilie\or mai\or
    iunie\or iulie\or august\or septembrie\or octombrie\or
    noiembrie\or decembrie\fi
    \space \number\year}}
```

`\extrasromanian` `\noextrasromanian` The macro `\extrasromanian` will perform all the extra definitions needed for the Romanian language. The macro `\noextrasromanian` is used to cancel the actions of `\extrasromanian` For the moment these macros are empty but they are defined for compatibility with the other language definition files.

```
\addto\extrasromanian{}
\addto\noextrasromanian{}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{romanian}
⟨/code⟩
```

## 40 The Danish language

The file `danish.dtx`[47] defines all the language definition macros for the Danish language.

For this language the character `"` is made active. In table 14 an overview is given of its purpose.

| | |
|---|---|
| `"\|` | disable ligature at this position. |
| `"-` | an explicit hyphen sign, allowing hyphenation in the rest of the word. |
| `""` | like `"-`, but producing no hyphen sign (for words that should break at some sign such as "entrada/salida." |
| ``"` `` | lowered double left quotes (looks like „) |
| `"'` | normal double right quotes |
| `"<` | for French left double quotes (similar to <<). |
| `">` | for French right double quotes (similar to >>). |

Table 14: The extra definitions made by `danish.ldf`

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
40.1 ⟨*code⟩
40.2 \LdfInit{danish}\captionsdanish
```

When this file is read as an option, i.e. by the `\usepackage` command, `danish` will be an 'unknown' language in which case we have to make it known. So we check for the existence of `\l@danish` to see whether we have to do something here.

```
40.3 \ifx\l@danish\@undefined
40.4     \@nopatterns{Danish}
40.5     \adddialect\l@danish0\fi
```

`\englishhyphenmins` This macro is used to store the correct values of the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`.

```
40.6 \providehyphenmins{\CurrentOption}{\@ne\tw@}
```

The next step consists of defining commands to switch to (and from) the Danish language.

`\captionsdanish` The macro `\captionsdanish` defines all strings used in the four standard documentclasses provided with LaTeX.

```
40.7 \addto\captionsdanish{%
40.8   \def\prefacename{Forord}%
40.9   \def\refname{Litteratur}%
40.10  \def\abstractname{Resum\'e}%
40.11  \def\bibname{Litteratur}%
```

[47]The file described in this section has version number v1.3p and was last revised on 2005/03/30. A contribution was made by Henning Larsen (`larsen@cernvm.cern.ch`)

```
  \def\chaptername{Kapitel}%
  \def\appendixname{Bilag}%
  \def\contentsname{Indhold}%
  \def\listfigurename{Figurer}%
  \def\listtablename{Tabeller}%
  \def\indexname{Indeks}%
  \def\figurename{Figur}%
  \def\tablename{Tabel}%
  \def\partname{Del}%
  \def\enclname{Vedlagt}%
  \def\ccname{Kopi til}%   or    Kopi sendt til
  \def\headtoname{Til}% in letter
  \def\pagename{Side}%
  \def\seename{Se}%
  \def\alsoname{Se ogs{\aa}}%
  \def\proofname{Bevis}%
  \def\glossaryname{Gloseliste}%
  }%
```

`\datedanish` The macro `\datedanish` redefines the command `\today` to produce Danish dates.

```
\def\datedanish{%
  \def\today{\number\day.~\ifcase\month\or
    januar\or februar\or marts\or april\or maj\or juni\or
    juli\or august\or september\or oktober\or november\or december\fi
    \space\number\year}}
```

`\extrasdanish` `\noextrasdanish` The macro `\extrasdanish` will perform all the extra definitions needed for the Danish language. The macro `\noextrasdanish` is used to cancel the actions of `\extrasdanish`.

Danish typesetting requires `\frenchspacing` to be in effect.

```
\addto\extrasdanish{\bbl@frenchspacing}
\addto\noextrasdanish{\bbl@nonfrenchspacing}
```

For Danish the `"` character is made active. This is done once, later on its definition may vary. Other languages in the same document may also use the `"` character for shorthands; we specify that the danish group of shorthands should be used.

```
\initiate@active@char{"}
\addto\extrasdanish{\languageshorthands{danish}}
\addto\extrasdanish{\bbl@activate{"}}
```

Don't forget to turn the shorthands off again.

```
\addto\noextrasdanish{\bbl@deactivate{"}}
```

First we define access to the low opening double quote and guillemets for quotations,

```
\declare@shorthand{danish}{"`}{%
  \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
\declare@shorthand{danish}{"'}{%
```

```
  \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
\declare@shorthand{danish}{"<}{%
  \textormath{\guillemotleft}{\mbox{\guillemotleft}}}
\declare@shorthand{danish}{">}{%
  \textormath{\guillemotright}{\mbox{\guillemotright}}}
```

then we define to be able to specify hyphenation breakpoints that behave a little different from \-.

```
\declare@shorthand{danish}{"-}{\nobreak-\bbl@allowhyphens}
\declare@shorthand{danish}{""}{\hskip\z@skip}
\declare@shorthand{danish}{"~}{\textormath{\leavevmode\hbox{-}}{-}}
\declare@shorthand{danish}{"=}{\nobreak-\hskip\z@skip}
```

And we want to have a shorthand for disabling a ligature.

```
\declare@shorthand{danish}{"|}{%
  \textormath{\discretionary{-}{}{\kern.03em}}{}}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{danish}
⟨/code⟩
```

# 41 The Icelandic language

## 41.1 Overview

The file `iceland.dtx`[48] defines all the language definition macros for the Icelandic language

Customization for the Icelandic language was made following several official and semiofficial publications [2, 3, 1, 6, 5]. These publications do not always agree and we indicate those instances.

For this language the character `"` is made active. In table 15 an overview is given of its purpose. The shorthands in table 15 can also be typeset by using the

| | |
|---|---|
| `"\|` | disable ligature at this position. |
| `"-` | an explicit hyphen sign, allowing hyphenation in the rest of the word. |
| `""` | like `"-`, but producing no hyphen sign (for compund words with hyphen, e.g. `x-""y`). |
| `"~` | for a compound word mark without a breakpoint. |
| `"=` | for a compound word mark with a breakpoint, allowing hyphenation in the composing words. |
| `"‘` | for Icelandic left double quotes (looks like „). |
| `"’` | for Icelandic right double quotes. |
| `">` | for Icelandic ‘french’ left double quotes (similar to >>). |
| `"<` | for Icelandic ‘french’ right double quotes (similar to <<). |
| `"o` | for old Icelandic ǫ |
| `"O` | for old Icelandic Ǫ |
| `"ó` | for old Icelandic ǫ́ |
| `"Ó` | for old Icelandic Ǫ́ |
| `"e` | for old Icelandic ę |
| `"E` | for old Icelandic Ę |
| `"é` | for old Icelandic ę́ |
| `"É` | for old Icelandic Ę́ |
| `\tala` | for typesetting numbers |
| `\grada` | for the ‘degree’ symbol |
| `\gradur` | for ‘degrees’, e.g. 5 °C |
| `\upp` | for textsuperscript |

Table 15: The shorthands and extra definitions made by `icelandic.ldf`

commands in table 16.

[48]The file described in this section has version number ? and was last revised on ?.

| | |
|---|---|
| `\ilqq` | for Icelandic left double quotes (looks like „). |
| `\irqq` | for Icelandic right double quotes (looks like “). |
| `\ilq` | for Icelandic left single quotes (looks like ‚). |
| `\irq` | for Icelandic right single quotes (looks like ‘). |
| `\iflqq` | for Icelandic ‘french’ left double quotes (similar to >>). |
| `\ifrqq` | for Icelandic ‘french’ right double quotes (similar to <<). |
| `\ifrq` | for Icelandic ‘french’ right single quotes (similar to <). |
| `\iflq` | for Icelandic ‘french’ left single quotes (similar to >). |
| `\dq` | the original quotes character ("). |
| `\oob` | for old Icelandic ǫ |
| `\Oob` | for old Icelandic Ǫ |
| `\ooob` | for old Icelandic ǫ́ |
| `\OOob` | for old Icelandic Ǫ́ |
| `\eob` | for old Icelandic ę |
| `\Eob` | for old Icelandic Ę |
| `\eeob` | for old Icelandic ę́ |
| `\EEob` | for old Icelandic Ę́ |

Table 16: Commands which produce quotes and old Icelandic diacritics, defined by `icelandic.ldf`

## References

[1] Alþingi. *Reglur um frágang þingskjala og prentun umræðna*, 1988.

[2] Auglýsing um greinarmerkjasetningu. Stj.tíð B, nr. 133/1974, 1974.

[3] Auglýsing um breyting auglýsingu nr. 132/1974 um íslenska stafsetningu. Stj.tíð B, nr. 261/1977, 1977.

[4] Einar Haugen, editor. *First Grammatical Treatise.* Longman, London, 2 edition, 1972.

[5] Staðlaráð Íslands og Fagráð í upplýsingatækni, Reykjavík. *Forstaðall FS 130:1997*, 1997.

[6] STRÍ Staðlaráð Íslands. *SI - kerfið*, 2 edition, 1994.

### 41.2 TEXnical details

When this file was read through the option icelandic we make it behave as if icelandic was specified.

```
41.1 \def\bbl@tempa{icelandic}
```

```
\ifx\CurrentOption\bbl@tempa
  \def\CurrentOption{icelandic}
\fi
```

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
⟨*code⟩
\LdfInit\CurrentOption{captions\CurrentOption}
```

When this file is read as an option, i.e., by the `\usepackage` command, `icelandic` will be an 'unknown' language, so we have to make it known. So we check for the existence of `\l@icelandic` to see whether we have to do something here.

```
\ifx\l@icelandic\@undefined
  \@nopatterns{Icelandic}
  \adddialect\l@icelandic0
\fi
```

`\if@Two@E` We will need a new 'if' : `\if@Two@E` is true if and only if LaTeX $2_\varepsilon$ is running *not* in compatibility mode. It is used in the definitions of the command `\tala` and `\upp`. The definition is somewhat complicated, due to the fact that `\if@compatibility` is not recognized as a `\if` in LaTeX-2.09 based formats.

```
\newif\if@Two@E \@Two@Etrue
\def\@FI@{\fi}
\ifx\@compatibilitytrue\@undefined
  \@Two@Efalse \def\@FI@{\relax}
\else
  \if@compatibility \@Two@Efalse \fi
\@FI@
```

`\extrasicelandic` `\noextrasicelandic` The macro `\extrasicelandic` will perform all the extra definitions needed for the Icelandic language. The macro `\noextrasicelandic` is used to cancel the actions of `\extrasicelandic`.

For Icelandic the `"` character is made active. This is done once, later on its definition may vary.

```
\initiate@active@char{"}
\@namedef{extras\CurrentOption}{%
  \languageshorthands{icelandic}}
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \bbl@activate{"}}
```

Don't forget to turn the shorthands off again.

```
\addto\noextrasicelandic{\bbl@deactivate{"}}
```

The icelandic hyphenation patterns can be used with `\lefthyphenmin` and `\righthyphenmin` set to 2.

```
\providehyphenmins{\CurrentOption}{\tw@\tw@}
```

The code above is necessary because we need an extra active character. This character is then used as indicated in table 16.

To be able to define the function of `"`, we first define a couple of 'support' macros.

## 41.3 Captionnames and date

The next step consists of defining the Icelandic equivalents for the LaTeX captionnames.

`\captionsicelandic` The macro `\captionsicelandic` will define all strings used used in the four standard document classes provided with LaTeX.

```
\@namedef{captions\CurrentOption}{%
  \def\prefacename{Form\'{a}li}%
  \def\refname{Heimildir}%
  \def\abstractname{\'{U}tdr\'{a}ttur}%
  \def\bibname{Heimildir}%
  \def\chaptername{Kafli}%
  \def\appendixname{Vi{\dh}auki}%
  \def\contentsname{Efnisyfirlit}%
  \def\listfigurename{Myndaskr\'{a}}%
  \def\listtablename{T\"{o}fluskr\'{a}}%
  \def\indexname{Atri{\dh}isor{\dh}askr\'{a}}%
  \def\figurename{Mynd}%
  \def\tablename{Tafla}%
  \def\partname{Hluti}%
  \def\enclname{Hj\'{a}lagt}%
  \def\ccname{Samrit}%
  \def\headtoname{Til:}% in letter
  \def\pagename{Bla{\dh}s\'{\i}{\dh}a}%
  \def\seename{Sj\'{a}}%
  \def\alsoname{Sj\'{a} einnig}%
  \def\proofname{S\"{o}nnun}%
  \def\glossaryname{Or{\dh}alisti}%
  }
```

`\dateicelandic` The macro `\dateicelandic` redefines the command `\today` to produce Icelandic dates.

```
\def\dateicelandic{%
  \def\today{\number\day.~\ifcase\month\or
    jan\'{u}ar\or febr\'{u}ar\or mars\or apr\'{\i}l\or ma\'{\i}\or
    j\'{u}n\'{\i}\or j\'{u}l\'{\i}\or \'{a}g\'{u}st\or september\or
    okt\'{o}ber\or n\'{o}vember\or desember\fi
    \space\number\year}}
```

## 41.4 Icelandic quotation marks

`\dq` We save the original double quote character in `\dq` to keep it available, the math accent `\"` can now be typed as `"`.

```
\begingroup \catcode`\"12
```

```
\def\x{\endgroup
  \def\@SS{\mathchar"7019 }
  \def\dq{"}}
\x
```

Now we can define the icelandic and icelandic 'french' quotes. The icelandic 'french' guillemets are the reverse of french guillemets. We define single icelandic 'french' quotes for compatibility. Shorthands are provided for a number of different quotation marks, which make them useable both outside and inside mathmode.

```
\let\ilq\grq
\let\irq\grq
\let\iflq\frq
\let\ifrq\flq
\let\ilqq\glqq
\let\irqq\grqq
\let\iflqq\frqq
\let\ifrqq\flqq
```

```
\declare@shorthand{icelandic}{"`}{\glqq}
\declare@shorthand{icelandic}{"'}{\grqq}
\declare@shorthand{icelandic}{">}{\frqq}
\declare@shorthand{icelandic}{"<}{\flqq}
```

and some additional commands:

```
\declare@shorthand{icelandic}{"-}{\nobreak\-\bbl@allowhyphens}
\declare@shorthand{icelandic}{"|}{%
  \textormath{\nobreak\discretionary{-}{}{\kern.03em}%
              \bbl@allowhyphens}{}}
\declare@shorthand{icelandic}{""}{\hskip\z@skip}
\declare@shorthand{icelandic}{"~}{\textormath{\leavevmode\hbox{-}}{-}}
\declare@shorthand{icelandic}{"=}{\nobreak-\hskip\z@skip}
```

## 41.5 Old Icelandic

In old Icelandic some letters have special diacritical marks, described for example in *First Grammatical Treatise* [4, 5]. We provide these in the `T1` encoding with the 'ogonek'. The ogonek is placed with the letters 'o', and 'O', 'ó' and 'Ó', 'e' and 'E', and 'é' and 'É'. Shorthands are provided for these as well.

The following code by Leszek Holenderski lifted from `polish.dtx` is designed to position the diacritics correctly for every font in every size. These macros need a few extra dimension variables.

```
\newdimen\pl@left
\newdimen\pl@down
\newdimen\pl@right
\newdimen\pl@temp
```

`\sob` The macro `\sob` is used to put the 'ogonek' in the right place.

```
\def\sob#1#2#3#4#5{%parameters: letter and fractions hl,ho,vl,vo
  \setbox0\hbox{#1}\setbox1\hbox{\k{}}\setbox2\hbox{p}%
\pl@right=#2\wd0 \advance\pl@right by-#3\wd1
\pl@down=#5\ht1 \advance\pl@down by-#4\ht0
\pl@left=\pl@right \advance\pl@left by\wd1
\pl@temp=-\pl@down \advance\pl@temp by\dp2 \dp1=\pl@temp
\leavevmode
\kern\pl@right\lower\pl@down\box1\kern-\pl@left #1}
```

\oob \Oob \ooob \OOob \eob \Eob \eeob \EEob

```
\DeclareTextCommand{\oob}{T1}{\sob {o}{.85}{0}{.04}{0}}
\DeclareTextCommand{\Oob}{T1}{\sob {O}{.7}{0}{0}{0}}
\DeclareTextCommand{\ooob}{T1}{\sob {ó}{.85}{0}{.04}{0}}
\DeclareTextCommand{\OOob}{T1}{\sob {Ó}{.7}{0}{0}{0}}
\DeclareTextCommand{\eob}{T1}{\sob {e}{1}{0}{.04}{0}}
\DeclareTextCommand{\Eob}{T1}{\sob {E}{1}{0}{.04}{0}}
\DeclareTextCommand{\eeob}{T1}{\sob {é}{1}{0}{.04}{0}}
\DeclareTextCommand{\EEob}{T1}{\sob {É}{1}{0}{.04}{0}}
```

```
\declare@shorthand{icelandic}{"o}{\oob}
\declare@shorthand{icelandic}{"O}{\Oob}
\declare@shorthand{icelandic}{"ó}{\ooob}
\declare@shorthand{icelandic}{"Ó}{\OOob}
\declare@shorthand{icelandic}{"e}{\eob}
\declare@shorthand{icelandic}{"E}{\Eob}
\declare@shorthand{icelandic}{"é}{\eeob}
\declare@shorthand{icelandic}{"É}{\EEob}
```

## 41.6 Formatting numbers

This section is lifted from `frenchb.dtx` by D. Flipo. In English the decimal part starts with a point and thousands should be separated by a comma: an approximation of $1000\pi$ should be inputed as `$3{,}141.592{,}653$` in math-mode and as `3,141.592,653` in text.

In Icelandic the decimal part starts with a comma and thousands should be separated by a space [1] or by a period [5]; we have the space. The above approximation of $1000\pi$ should be inputed as `$3\;141{,}592\;653$` in math-mode and as something like `3~141,592~653` in text. Braces are mandatory around the comma in math-mode, the reason is mentioned in the TEXbook p. 134: the comma is of type `\mathpunct` (thus normally followed by a space) while the point is of type `\mathord` (no space added).

Thierry Bouche suggested that a second type of comma, of type `\mathord` would be useful in math-mode, and proposed to introduce a command (named `\decimalsep` in this package), the expansion of which would depend on the current language.

Vincent Jalby suggested a command `\nombre` to conveniently typeset numbers: inputting `\nombre{3141,592653}` either in text or in math-mode will format this number properly according to the current language (Icelandic or non-Icelandic). We use `\nombre` to define command `\tala` in Icelandic.

`\tala` accepts an optional argument which happens to be useful with the extension 'dcolumn', it specifies the decimal separator used in the *source code*:

```
\newcolumntype{d}{D{,}{\decimalsep}{-1}}
   \begin{tabular}{d}\hline
     3,14 \\
     \tala[,]{123,4567} \\
     \tala[,]{9876,543}\\\hline
   \end{tabular}
```

will print a column of numbers aligned on the decimal point (comma or point depending on the current language), each slice of 3 digits being separated by a space or a comma according to the current language.

`\decimalsep` `\thousandsep` We need a internal definition, valid in both text and math-mode, for the comma (`\@comma@`) and another one for the unbreakable fixed length space (no glue) used in Icelandic (`\f@thousandsep`).

The commands `\decimalsep` and `\thousandsep` get default definitions (for the English language) when `icelandic` is loaded; these definitions will be updated when the current language is switched to or from Icelandic.

```
\mathchardef\m@comma="013B \def\@comma@{\ifmmode\m@comma\else,\fi}
\def\f@thousandsep{\ifmmode\mskip5.5mu\else\penalty\@M\kern.3em\fi}
\newcommand{\decimalsep}{.}  \newcommand{\thousandsep}{\@comma@}
\expandafter\addto\csname extras\CurrentOption\endcsname{%
             \def\decimalsep{\@comma@}%
             \def\thousandsep{\f@thousandsep}}
\expandafter\addto\csname noextras\CurrentOption\endcsname{%
             \def\decimalsep{.}%
             \def\thousandsep{\@comma@}}
```

`\tala` The decimal separator used when *inputing* a number with `\tala` *has to be a comma.* `\tala` splits the inputed number into two parts: what comes before the first comma will be formatted by `\@integerpart` while the rest (if not empty) will be formatted by `\@decimalpart`. Both parts, once formatted separately will be merged together with between them, either the decimal separator `\decimalsep` or (in LaTeX $2_\varepsilon$ *only*) the optional argument of `\tala`.

```
\if@Two@E
  \newcommand{\tala}[2][\decimalsep]{%
          \def\@decimalsep{#1}\@tala#2\@empty,\@empty,\@nil}
\else
  \newcommand{\tala}[1]{%
          \def\@decimalsep{\decimalsep}\@tala#1\@empty,\@empty,\@nil}
\fi
\def\@tala#1,#2,#3\@nil{%
        \ifx\@empty#2%
          \@integerpart{#1}%
        \else
          \@integerpart{#1}\@decimalsep\@decimalpart{#2}%
        \fi}
```

The easiest bit is the decimal part: We attempt to read the first four digits of the decimal part, if it has less than 4 digits, we just have to print them, otherwise `\thousandsep` has to be appended after the third digit, and the algorithm is applied recursively to the rest of the decimal part.

```
\def\@decimalpart#1{\@@decimalpart#1\@empty\@empty\@empty}
\def\@@decimalpart#1#2#3#4{#1#2#3%
  \ifx\@empty#4%
  \else
    \thousandsep\expandafter\@@decimalpart\expandafter#4%
  \fi}
```

Formatting the integer part is more difficult because the slices of 3 digits start from the *bottom* while the number is read from the top! This (tricky) code is borrowed from David Carlisle's comma.sty.

```
\def\@integerpart#1{\@@integerpart{}#1\@empty\@empty\@empty}
\def\@@integerpart#1#2#3#4{%
  \ifx\@empty#2%
    \@addthousandsep#1\relax
  \else
    \ifx\@empty#3%
      \@addthousandsep\@empty\@empty#1#2\relax
    \else
      \ifx\@empty#4%
        \@addthousandsep\@empty#1#2#3\relax
      \else
        \@@integerpartafterfi{#1#2#3#4}%
      \fi
    \fi
  \fi}
\def\@@integerpartafterfi#1\fi\fi\fi{\fi\fi\fi\@@integerpart{#1}}
\def\@addthousandsep#1#2#3#4{#1#2#3%
  \if#4\relax
  \else
    \thousandsep\expandafter\@addthousandsep\expandafter#4%
  \fi}
```

## 41.7 Extra utilities

We now provide the Icelandic user with some extra utilities.

`\upp` — `\upp` is for typesetting superscripts. `\upp` relies on

`\upp@size` — The internal macro `\upp@size` holds the size at which the superscript will be typeset. The reason for this is that we have to specify it differently for different formats.

```
\ifx\sevenrm\@undefined
  \ifx\@ptsize\@undefined
    \let\upp@size\small
  \else
    \ifx\selectfont\@undefined
```

In this case the format is the original LaTeX-2.09:

```
      \ifcase\@ptsize
        \let\upp@size\ixpt\or
        \let\upp@size\xpt\or
        \let\upp@size\xipt
      \fi
```

When `\selectfont` is defined we probably have NFSS available:

```
    \else
      \ifcase\@ptsize
        \def\upp@size{\fontsize\@ixpt{10pt}\selectfont}\or
        \def\upp@size{\fontsize\@xpt{11pt}\selectfont}\or
        \def\upp@size{\fontsize\@xipt{12pt}\selectfont}
      \fi
    \fi
  \fi
\else
```

If we end up here it must be a plain based TeX format, so:

```
    \let\upp@size\sevenrm
\fi
```

Now we can define `\upp`. When LaTeX $2_\varepsilon$ runs in compatibility mode (LaTeX-2.09 emulation), `\textsuperscript` is also defined, but does no good job, so we give two different definitions for `\upp` using `\if@Two@E`.

```
\if@Two@E
  \DeclareRobustCommand*{\upp}[1]{\textsuperscript{#1}}
\else
  \DeclareRobustCommand*{\upp}[1]{%
    \leavevmode\raise1ex\hbox{\upp@size#1}}
\fi
```

Some definitions for special characters. `\grada` needs a special treatment: it is `\char6` in T1-encoding and `\char23` in OT1-encoding.

```
\ifx\fmtname\LaTeXeFmtName
  \DeclareTextSymbol{\grada}{T1}{6}
  \DeclareTextSymbol{\grada}{OT1}{23}
\else
  \def\T@one{T1}
  \ifx\f@encoding\T@one
    \newcommand{\grada}{\char6}
  \else
    \newcommand{\grada}{\char23}
  \fi
\fi
```

`\gradur` Macro for typesetting the abbreviation for 'degrees' (as in 'degrees Celsius'). As the bounding box of the character 'degree' has *very* different widths in CMR/DC and PostScript fonts, we fix the width of the bounding box of `\gradur` to 0.3 em, this lets the symbol ‘degree’ stick to the preceding (e.g., `45\gradur`) or following character (e.g., `20~\gradur C`).

```
\DeclareRobustCommand*{\gradur}{%
                            \leavevmode\hbox to 0.3em{\hss\grada\hss}}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish\CurrentOption
⟨/code⟩
```

## 42 The Norwegian language

The file `norsk.dtx`[49] defines all the language definition macros for the Norwegian language as well as for an alternative variant 'nynorsk' of this language.

For this language the character `"` is made active. In table 17 an overview is given of its purpose.

| | |
|---|---|
| `"ff` | for `ff` to be hyphenated as `ff-f`, this is also implemented for b, d, f, g, l, m, n, p, r, s, and t. (`o"ppussing`) |
| `"ee` | Hyphenate `"ee` as `\'e-e`. (`komit"een`) |
| `"-` | an explicit hyphen sign, allowing hyphenation in the composing words. Use this for compound words when the hyphenation patterns fail to hyphenate properly. (`alpin"-anlegg`) |
| `"\|` | Like `"-`, but inserts 0.03em space. Use it if the compound point is spanned by a ligature. (`hoff"\|intriger`) |
| `""` | Like `"-`, but producing no hyphen sign. (`i""g\aa{}r`) |
| `"~` | Like `-`, but allows no hyphenation at all. (`E"~cup`) |
| `"=` | Like `-`, but allowing hyphenation in the composing words. (`marksistisk"=leninistisk`) |
| `"<` | for French left double quotes (similar to <<). |
| `">` | for French right double quotes (similar to >>). |

Table 17: The extra definitions made by `norsk.sty`

Rune Kleveland distributes a Norwegian dictionary for ispell (570000 words). It can be found at `http://www.uio.no/~runekl/dictionary.html`.

This dictionary supports the spellings `spi"sslede` for 'spisslede' (hyphenated spiss-slede) and other such words, and also suggest the spelling `spi"sslede` for 'spisslede' and 'spissslede'.

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
⟨*code⟩
\LdfInit\CurrentOption{captions\CurrentOption}
```

When this file is read as an option, i.e. by the `\usepackage` command, `norsk` will be an 'unknown' language in which case we have to make it known. So we check for the existence of `\l@norsk` to see whether we have to do something here.

```
\ifx\l@norsk\@undefined
    \@nopatterns{Norsk}
    \adddialect\l@norsk0\fi
```

[49]The file described in this section has version number v2.0h and was last revised on 2005/03/30. Contributions were made by Haavard Helstrup (`HAAVARD@CERNVM`) and Alv Kjetil Holme (`HOLMEA@CERNVM`); the 'nynorsk' variant has been supplied by Per Steinar Iversen `iversen@vxcern.cern.ch`) and Terje Engeset Petterst (`TERJEEP@VSFYS1.FI.UIB.NO`); the shorthand definitions were provided by Rune Kleveland (`runekl@math.uio.no`).

\norskhyphenmins Some sets of Norwegian hyphenation patterns can be used with `\lefthyphenmin` set to 1 and `\righthyphenmin` set to 2, but the most common set `nohyph.tex` can't. So we use `\lefthyphenmin=2` by default.

```
\providehyphenmins{\CurrentOption}{\tw@\tw@}
```

Now we have to decide which version of the captions should be made available. This can be done by checking the contents of `\CurrentOption`.

```
\def\bbl@tempa{norsk}
\ifx\CurrentOption\bbl@tempa
```

The next step consists of defining commands to switch to (and from) the Norwegian language.

\captionsnorsk The macro `\captionsnorsk` defines all strings used in the four standard documentclasses provided with LaTeX.

```
  \def\captionsnorsk{%
    \def\prefacename{Forord}%
    \def\refname{Referanser}%
    \def\abstractname{Sammendrag}%
    \def\bibname{Bibliografi}%     or Litteraturoversikt
    %                              or Litteratur or Referanser
    \def\chaptername{Kapittel}%
    \def\appendixname{Tillegg}%    or Appendiks
    \def\contentsname{Innhold}%
    \def\listfigurename{Figurer}%  or Figurliste
    \def\listtablename{Tabeller}%  or Tabelliste
    \def\indexname{Register}%
    \def\figurename{Figur}%
    \def\tablename{Tabell}%
    \def\partname{Del}%
    \def\enclname{Vedlegg}%
    \def\ccname{Kopi sendt}%
    \def\headtoname{Til}% in letter
    \def\pagename{Side}%
    \def\seename{Se}%
    \def\alsoname{Se ogs\aa{}}%
    \def\proofname{Bevis}%
    \def\glossaryname{Ordliste}%
    }
\else
```

For the 'nynorsk' version of these definitions we just add a "dialect".

```
  \adddialect\l@nynorsk\l@norsk
```

\captionsnynorsk The macro `\captionsnynorsk` defines all strings used in the four standard documentclasses provided with LaTeX, but using a different spelling than in the command `\captionsnorsk`.

```
  \def\captionsnynorsk{%
    \def\prefacename{Forord}%
    \def\refname{Referansar}%
    \def\abstractname{Samandrag}%
    \def\bibname{Litteratur}%     or Litteraturoversyn
    %                              or Referansar
    \def\chaptername{Kapittel}%
    \def\appendixname{Tillegg}%   or Appendiks
    \def\contentsname{Innhald}%
    \def\listfigurename{Figurar}% or Figurliste
    \def\listtablename{Tabellar}% or Tabelliste
    \def\indexname{Register}%
    \def\figurename{Figur}%
    \def\tablename{Tabell}%
    \def\partname{Del}%
    \def\enclname{Vedlegg}%
    \def\ccname{Kopi til}%
    \def\headtoname{Til}% in letter
    \def\pagename{Side}%
    \def\seename{Sj\aa{}}%
    \def\alsoname{Sj\aa{} \`{o}g}%
    \def\proofname{Bevis}%
    \def\glossaryname{Ordliste}%
    }
\fi
```

`\datenorsk` The macro `\datenorsk` redefines the command `\today` to produce Norwegian dates.

```
\@namedef{date\CurrentOption}{%
  \def\today{\number\day.~\ifcase\month\or
    januar\or februar\or mars\or april\or mai\or juni\or
    juli\or august\or september\or oktober\or november\or desember
    \fi
    \space\number\year}}
```

`\extrasnorsk` `\extrasnynorsk` The macro `\extrasnorsk` will perform all the extra definitions needed for the Norwegian language. The macro `\noextrasnorsk` is used to cancel the actions of `\extrasnorsk`.

Norwegian typesetting requires `\frencspacing` to be in effect.

```
\@namedef{extras\CurrentOption}{\bbl@frenchspacing}
\@namedef{noextras\CurrentOption}{\bbl@nonfrenchspacing}
```

For Norsk the `"` character is made active. This is done once, later on its definition may vary.

```
\initiate@active@char{"}
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \languageshorthands{norsk}}
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \bbl@activate{"}}
```

Don't forget to turn the shorthands off again.

```
\expandafter\addto\csname noextras\CurrentOption\endcsname{%
  \bbl@deactivate{"}}
```

The code above is necessary because we need to define a number of shorthand commands. These sharthand commands are then used as indicated in table 17.

To be able to define the function of ", we first define a couple of 'support' macros.

\dq We save the original double quote character in \dq to keep it available, the math accent \" can now be typed as ".

```
\begingroup \catcode`\"12
\def\x{\endgroup
  \def\@SS{\mathchar"7019 }
  \def\dq{"}}
\x
```

Now we can define the discretionary shorthand commands. The number of words where such hyphenation is required is for each character

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| b | d | f | g | k | l | n | p | r | s | t |
| 4 | 4 | 15 | 3 | 43 | 30 | 8 | 12 | 1 | 33 | 35 |

taken from a list of 83000 ispell-roots.

```
\declare@shorthand{norsk}{"b}{\textormath{\bbl@disc b{bb}}{b}}
\declare@shorthand{norsk}{"B}{\textormath{\bbl@disc B{BB}}{B}}
\declare@shorthand{norsk}{"d}{\textormath{\bbl@disc d{dd}}{d}}
\declare@shorthand{norsk}{"D}{\textormath{\bbl@disc D{DD}}{D}}
\declare@shorthand{norsk}{"e}{\textormath{\bbl@disc e{\'e}}{}}
\declare@shorthand{norsk}{"E}{\textormath{\bbl@disc E{\'E}}{}}
\declare@shorthand{norsk}{"F}{\textormath{\bbl@disc F{FF}}{F}}
\declare@shorthand{norsk}{"g}{\textormath{\bbl@disc g{gg}}{g}}
\declare@shorthand{norsk}{"G}{\textormath{\bbl@disc G{GG}}{G}}
\declare@shorthand{norsk}{"k}{\textormath{\bbl@disc k{kk}}{k}}
\declare@shorthand{norsk}{"K}{\textormath{\bbl@disc K{KK}}{K}}
\declare@shorthand{norsk}{"l}{\textormath{\bbl@disc l{ll}}{l}}
\declare@shorthand{norsk}{"L}{\textormath{\bbl@disc L{LL}}{L}}
\declare@shorthand{norsk}{"n}{\textormath{\bbl@disc n{nn}}{n}}
\declare@shorthand{norsk}{"N}{\textormath{\bbl@disc N{NN}}{N}}
\declare@shorthand{norsk}{"p}{\textormath{\bbl@disc p{pp}}{p}}
\declare@shorthand{norsk}{"P}{\textormath{\bbl@disc P{PP}}{P}}
\declare@shorthand{norsk}{"r}{\textormath{\bbl@disc r{rr}}{r}}
\declare@shorthand{norsk}{"R}{\textormath{\bbl@disc R{RR}}{R}}
\declare@shorthand{norsk}{"s}{\textormath{\bbl@disc s{ss}}{s}}
\declare@shorthand{norsk}{"S}{\textormath{\bbl@disc S{SS}}{S}}
\declare@shorthand{norsk}{"t}{\textormath{\bbl@disc t{tt}}{t}}
\declare@shorthand{norsk}{"T}{\textormath{\bbl@disc T{TT}}{T}}
```

We need to treat "f a bit differently in order to preserve the ff-ligature.

```
\declare@shorthand{norsk}{"f}{\textormath{\bbl@discff}{f}}
\def\bbl@discff{\penalty\@M
```

```
  \afterassignment\bbl@insertff \let\bbl@nextff= }
\def\bbl@insertff{%
  \if f\bbl@nextff
    \expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi
  {\relax\discretionary{ff-}{f}{ff}\allowhyphens}{f\bbl@nextff}}
\let\bbl@nextff=f
```

We now define the French double quotes and some commands concerning hyphenation:

```
\declare@shorthand{norsk}{"<}{\flqq}
\declare@shorthand{norsk}{">}{\frqq}
\declare@shorthand{norsk}{"-}{\penalty\@M\-\bbl@allowhyphens}
\declare@shorthand{norsk}{"|}{%
  \textormath{\penalty\@M\discretionary{-}{}{\kern.03em}%
              \allowhyphens}{}}
\declare@shorthand{norsk}{""}{\hskip\z@skip}
\declare@shorthand{norsk}{"~}{\textormath{\leavevmode\hbox{-}}{-}}
\declare@shorthand{norsk}{"=}{\penalty\@M-\hskip\z@skip}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish\CurrentOption
⟨/code⟩
```

## 43 The Swedish language

The file `swedish.dtx`[50] defines all the language-specific macros for the Swedish language. This file has borrowed heavily from `finnish.dtx` and `germanb.dtx`.

For this language the character `"` is made active. In table 18 an overview is given of its purpose. The vertical placement of the "umlaut" in some letters can be controlled this way.

| | |
|---|---|
| `"a` | Gives ä, also implemented for `"A`, `"o` and `"O`. |
| `"w`, `"W` | gives å and Å. |
| `"ff` | for `ff` to be hyphenated as `ff-f`. Used for compound words, such as `stra"ffånge`, which should be hyphenated as `straff-fånge`. This is also implemented for b, d, f, g, l, m, n, p, r, s, and t. |
| `"\|` | disable ligature at this position. This should be used for compound words, such as "`stra"ffinrättning`", which should not have the ligature "ffi". |
| `"-` | an explicit hyphen sign, allowing hyphenation in the rest of the word, such as e. g. in "x`"-`axeln". |
| `""` | like `"-`, but producing no hyphen sign (for words that should break at some sign such as `och/""eller`). |
| `"~` | for an explicit hyphen without a breakpoint; useful for expressions such as "2`"~`3 veckor" where no linebreak is desirable. |
| `"=` | an explicit hyphen sign allowing subsequent hyphenation, for expressions such as "studiebidrag och -lån". |
| `\-` | like the old `\-`, but allowing hyphenation in the rest of the word. |

Table 18: The extra definitions made by `swedish.sty`

Two variations for formatting of dates are added. `\datesymd` makes `\today` output dates formatted as YYYY-MM-DD, which is commonly used in Sweden today. `\datesdmy` formats the date as D/M YYYY, which is also very common in Sweden. These commands should be issued after `\begindocument`.

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
43.1 ⟨*code⟩
43.2 \LdfInit{swedish}\captionsswedish
```

When this file is read as an option, i.e. by the `\usepackage` command, `swedish` will be an 'unknown' language in which case we have to make it known. So we

[50]The file described in this section has version number v2.3d and was last revised on 2005/03/31. Contributions were made by Sten Hellman (`HELLMAN@CERNVM.CERN.CH`) and Erik Östhols (`erik_osthols@yahoo.com`).

check for the existence of `\l@swedish` to see whether we have to do something here.

```
\ifx\l@swedish\@undefined
    \@nopatterns{Swedish}
    \adddialect\l@swedish0\fi
```

The next step consists of defining commands to switch to the Swedish language. The reason for this is that a user might want to switch back and forth between languages.

`\captionsswedish` The macro `\captionsswedish` defines all strings used in the four standard documentclasses provided with LaTeX.

```
\addto\captionsswedish{%
  \def\prefacename{F\"orord}%
  \def\refname{Referenser}%
  \def\abstractname{Sammanfattning}%
  \def\bibname{Litteraturf\"orteckning}%
  \def\chaptername{Kapitel}%
  \def\appendixname{Bilaga}%
  \def\contentsname{Inneh\csname aa\endcsname ll}%
  \def\listfigurename{Figurer}%
  \def\listtablename{Tabeller}%
  \def\indexname{Sakregister}%
  \def\figurename{Figur}%
  \def\tablename{Tabell}%
  \def\partname{Del}%

  \def\enclname{Bil.}%
  \def\ccname{Kopia f\"or k\"annedom}%
  \def\headtoname{Till}% in letter
  \def\pagename{Sida}%
  \def\seename{se}%

  \def\alsoname{se \"aven}%

  \def\proofname{Bevis}%
  \def\glossaryname{Ordlista}%
  }%
```

`\dateswedish` The macro `\dateswedish` redefines the command `\today` to produce Swedish dates.

```
\def\dateswedish{%
  \def\today{%
    \number\day~\ifcase\month\or
    januari\or februari\or mars\or april\or maj\or juni\or
    juli\or augusti\or september\or oktober\or november\or
    december\fi
    \space\number\year}}
```

`\datesymd` The macro `\datesymd` redefines the command `\today` to produce dates in the format YYYY-MM-DD, common in Sweden.

```
\def\datesymd{%
  \def\today{\number\year-\two@digits\month-\two@digits\day}%
  }
```

`\datesdmy` The macro `\datesdmy` redefines the command `\today` to produce Swedish dates in the format DD/MM YYYY, also common in Sweden.

```
\def\datesdmy{%
  \def\today{\number\day/\number\month\space\number\year}%
  }
```

`\swedishhyphenmins` The swedish hyphenation patterns can be used with `\lefthyphenmin` set to 2 and `\righthyphenmin` set to 2.

```
\providehyphenmins{swedish}{\tw@\tw@}
```

`\extrasswedish` `\noextrasswedish` The macro `\extrasswedish` performs all the extra definitions needed for the Swedish language. The macro `\noextrasswedish` is used to cancel the actions of `\extrasswedish`.

For Swedish texts `\frenchspacing` should be in effect. We make sure this is the case and reset it if necessary.

```
\addto\extrasswedish{\bbl@frenchspacing}
\addto\noextrasswedish{\bbl@nonfrenchspacing}
```

For Swedish the `"` character is made active. This is done once, later on its definition may vary.

```
\initiate@active@char{"}
\addto\extrasswedish{\languageshorthands{swedish}}
\addto\extrasswedish{\bbl@activate{"}}
```

Don't forget to turn the shorthands off again.

```
\addto\noextrasswedish{\bbl@deactivate{"}}
```

The "umlaut" accent macro `\"` is changed to lower the "umlaut" dots. The redefinition is done with the help of `\umlautlow`.

```
\addto\extrasswedish{\babel@save\"\umlautlow}
\addto\noextrasswedish{\umlauthigh}
```

The code above is necessary because we need an extra active character. This character is then used as indicated in table 18.

To be able to define the function of `"`, we first define a couple of 'support' macros.

`\dq` We save the original double quote character in `\dq` to keep it available, the math accent `\"` can now be typed as `"`.

```
\begingroup \catcode`\"12
\def\x{\endgroup
  \def\@SS{\mathchar"7019 }
  \def\dq{"}}
\x
```

Now we can define the doublequote macros: the umlauts and å.

```
\declare@shorthand{swedish}{"w}{\textormath{{\aa}\allowhyphens}{\ddot w}}
\declare@shorthand{swedish}{"a}{\textormath{\"{a}\allowhyphens}{\ddot a}}
\declare@shorthand{swedish}{"o}{\textormath{\"{o}\allowhyphens}{\ddot o}}
\declare@shorthand{swedish}{"W}{\textormath{{\AA}\allowhyphens}{\ddot W}}
\declare@shorthand{swedish}{"A}{\textormath{\"{A}\allowhyphens}{\ddot A}}
\declare@shorthand{swedish}{"O}{\textormath{\"{O}\allowhyphens}{\ddot O}}
```

discretionary commands

```
\declare@shorthand{swedish}{"b}{\textormath{\bbl@disc b{bb}}{b}}
\declare@shorthand{swedish}{"B}{\textormath{\bbl@disc B{BB}}{B}}
\declare@shorthand{swedish}{"d}{\textormath{\bbl@disc d{dd}}{d}}
\declare@shorthand{swedish}{"D}{\textormath{\bbl@disc D{DD}}{D}}
\declare@shorthand{swedish}{"f}{\textormath{\bbl@disc f{ff}}{f}}
\declare@shorthand{swedish}{"F}{\textormath{\bbl@disc F{FF}}{F}}
\declare@shorthand{swedish}{"g}{\textormath{\bbl@disc g{gg}}{g}}
\declare@shorthand{swedish}{"G}{\textormath{\bbl@disc G{GG}}{G}}
\declare@shorthand{swedish}{"l}{\textormath{\bbl@disc l{ll}}{l}}
\declare@shorthand{swedish}{"L}{\textormath{\bbl@disc L{LL}}{L}}
\declare@shorthand{swedish}{"m}{\textormath{\bbl@disc m{mm}}{m}}
\declare@shorthand{swedish}{"M}{\textormath{\bbl@disc M{MM}}{M}}
\declare@shorthand{swedish}{"n}{\textormath{\bbl@disc n{nn}}{n}}
\declare@shorthand{swedish}{"N}{\textormath{\bbl@disc N{NN}}{N}}
\declare@shorthand{swedish}{"p}{\textormath{\bbl@disc p{pp}}{p}}
\declare@shorthand{swedish}{"P}{\textormath{\bbl@disc P{PP}}{P}}
\declare@shorthand{swedish}{"r}{\textormath{\bbl@disc r{rr}}{r}}
\declare@shorthand{swedish}{"R}{\textormath{\bbl@disc R{RR}}{R}}
\declare@shorthand{swedish}{"s}{\textormath{\bbl@disc s{ss}}{s}}
\declare@shorthand{swedish}{"S}{\textormath{\bbl@disc S{SS}}{S}}
\declare@shorthand{swedish}{"t}{\textormath{\bbl@disc t{tt}}{t}}
\declare@shorthand{swedish}{"T}{\textormath{\bbl@disc T{TT}}{T}}
```

and some additional commands:

```
\declare@shorthand{swedish}{"-}{\nobreak-\bbl@allowhyphens}
\declare@shorthand{swedish}{"|}{%
  \textormath{\nobreak\discretionary{-}{}{\kern.03em}%
              \bbl@allowhyphens}{}}
\declare@shorthand{swedish}{""}{\hskip\z@skip}
\declare@shorthand{swedish}{"~}{%
  \textormath{\leavevmode\hbox{-}\bbl@allowhyphens}{-}}
\declare@shorthand{swedish}{"=}{\hbox{-}\allowhyphens}
```

`\-` Redefinition of `\-`. The new version of `\-` should indicate an extra hyphenation position, while allowing other hyphenation positions to be generated automatically. The standard behaviour of TEX in this respect is very unfortunate for languages such as Dutch, Finnish, German and Swedish, where long compound words are quite normal and all one needs is a means to indicate an extra hyphenation position on top of the ones that TEX can generate from the hyphenation patterns.

```
\addto\extrasswedish{\babel@save\-}
\addto\extrasswedish{\def\-{\allowhyphens
```

```
                                   \discretionary{-}{}{}\allowhyphens}}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{swedish}
⟨/code⟩
```

## 44 The North Sami language

The file `samin.dtx`[51] defines all the language definition macros for the North Sami language.

Several Sami dialects/languages are spoken in Finland, Norway, Sweden and on the Kola Peninsula (Russia). The alphabets differ, so there will eventually be a need for more `.dtx` files for e.g. Lule and South Sami. Hence the name `samin.dtx` (and not `sami.dtx` or the like) in the North Sami case.

There are currently no hyphenation patterns available for the North Sami language, but you might consider using the patterns for Finnish (`fi8hyph.tex`), Norwegian (`nohyph.tex`) or Swedish (`sehyph.tex`). Add a line for the `samin` language to the `language.dat` file, and rebuild the LaTeX format file. See the documentation for your LaTeX distribution.

A note on writing North Sami in LaTeX: The TI encoding and EC fonts do not include the T WITH STROKE letter, which you will need a workaround for. My suggestion is to place the lines

```
\newcommand{\tx}{\mbox{t\hspace{-.35em}-}}
\newcommand{\txx}{\mbox{T\hspace{-.5em}-}}
```

in the preamble of your documents. They define the commands
`\txx{}` for LATIN CAPITAL LETTER T WITH STROKE and
`\tx{}` for LATIN SMALL LETTER T WITH STROKE.

### 44.1 The code of `samin.dtx`

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
⟨*code⟩
\LdfInit{samin}{captionssamin}
```

When this file is read as an option, i.e. by the `\usepackage` command, `samin` could be an 'unknown' language in which case we have to make it known. So we check for the existence of `\l@samin` to see whether we have to do something here.

```
\ifx\undefined\l@samin
  \@nopatterns{Samin}
  \adddialect\l@samin0\fi
```

The next step consists of defining commands to switch to (and from) the North Sami language.

`\saminhyphenmins` This macro is used to store the correct values of the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`.

```
\providehyphenmins{samin}{\tw@\tw@}
```

[51]The file described in this section has version number v1.0c and was last revised on 2004/02/20. It was written by Regnor Jernsletten, (`Regnor.Jernsletten@sami.uit.no`) or (`Regnor.Jernsletten@eunet.no`).

\captionssamin The macro `\captionssamin` defines all strings used in the four standard documentclasses provided with LaTeX.

```
\def\captionssamin{%
  \def\prefacename{Ovdas\'atni}%
  \def\refname{\v Cujuhusat}%
  \def\abstractname{\v Coahkk\'aigeassu}%
  \def\bibname{Girjj\'ala\v svuohta}%
  \def\chaptername{Kapihttal}%
  \def\appendixname{\v Cuovus}%
  \def\contentsname{Sisdoallu}%
  \def\listfigurename{Govvosat}%
  \def\listtablename{Tabeallat}%
  \def\indexname{Registtar}%
  \def\figurename{Govus}%
  \def\tablename{Tabealla}%
  \def\partname{Oassi}%
  \def\enclname{Mielddus}%
  \def\ccname{Kopia s\'addejuvvon}%
  \def\headtoname{Vuost\'aiv\'aldi}%
  \def\pagename{Siidu}%
  \def\seename{geah\v ca}%
  \def\alsoname{geah\v ca maidd\'ai}%
  \def\proofname{Duo\dj{}a\v stus}%
  \def\glossaryname{S\'atnelistu}%
}%
```

\datesamin The macro `\datesamin` redefines the command `\today` to produce North Sami dates.

```
\def\datesamin{%
  \def\today{\ifcase\month\or
    o\dj{}\dj{}ajagem\'anu\or
    guovvam\'anu\or
    njuk\v cam\'anu\or
    cuo\ng{}om\'anu\or
    miessem\'anu\or
    geassem\'anu\or
    suoidnem\'anu\or
    borgem\'anu\or
    \v cak\v cam\'anu\or
    golggotm\'anu\or
    sk\'abmam\'anu\or
    juovlam\'anu\fi
    \space\number\day.~b.\space\number\year}%
}%
```

\extrassamin
\noextrassamin The macro `\extrassamin` will perform all the extra definitions needed for the North Sami language. The macro `\noextrassamin` is used to cancel the actions of `\extrassamin`. For the moment these macros are empty but they are defined for compatibility with the other language definition files.

```
\addto\extrassamin{}
\addto\noextrassamin{}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{samin}
⟨/code⟩
```

## 45 The Finnish language

The file `finnish.dtx`[52] defines all the language definition macros for the Finnish language.

For this language the character `"` is made active. In table 19 an overview is given of its purpose.

| | |
|---|---|
| `"\|` | disable ligature at this position. |
| `"-` | an explicit hyphen sign, allowing hyphenation in the rest of the word. |
| `"=` | an explicit hyphen sign for expressions such as “pakastekaapit ja -arkut”. |
| `""` | like `"-`, but producing no hyphen sign (for words that should break at some sign such as “entrada/salida.” |
| `"‘` | lowered double left quotes (looks like „) |
| `"’` | normal double right quotes |
| `"<` | for French left double quotes (similar to <<). |
| `">` | for French right double quotes (similar to >>). |
| `\-` | like the old `\-`, but allowing hyphenation in the rest of the word. |

Table 19: The extra definitions made by `finnish.ldf`

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
45.1 ⟨*code⟩
45.2 \LdfInit{finnish}\captionsfinnish
```

When this file is read as an option, i.e. by the `\usepackage` command, `finnish` will be an ‘unknown’ language in which case we have to make it known. So we check for the existence of `\l@finnish` to see whether we have to do something here.

```
45.3 \ifx\l@finnish\@undefined
45.4     \@nopatterns{Finnish}
45.5     \adddialect\l@finnish0\fi
```

The next step consists of defining commands to switch to the Finnish language. The reason for this is that a user might want to switch back and forth between languages.

`\captionsfinnish` The macro `\captionsfinnish` defines all strings used in the four standard documentclasses provided with LaTeX.

```
45.6 \addto\captionsfinnish{%
45.7   \def\prefacename{Esipuhe}%
45.8   \def\refname{Viitteet}%
```

[52]The file described in this section has version number v1.3p and was last revised on 2005/03/30. A contribution was made by Mikko KANERVA (`KANERVA@CERNVM`) and Keranen Reino (`KERANEN@CERNVM`).

```
\def\abstractname{Tiivistelm\"a}
\def\bibname{Kirjallisuutta}%
\def\chaptername{Luku}%
\def\appendixname{Liite}%
\def\contentsname{Sis\"alt\"o}%   /* Could be "Sis\"allys" as well */
\def\listfigurename{Kuvat}%
\def\listtablename{Taulukot}%
\def\indexname{Hakemisto}%
\def\figurename{Kuva}%
\def\tablename{Taulukko}%
\def\partname{Osa}%
\def\enclname{Liitteet}%
\def\ccname{Jakelu}%
\def\headtoname{Vastaanottaja}%
\def\pagename{Sivu}%
\def\seename{katso}%
\def\alsoname{katso my\"os}%
\def\proofname{Todistus}%
\def\glossaryname{Sanasto}%
}%
```

`\datefinnish` The macro `\datefinnish` redefines the command `\today` to produce Finnish dates.

```
\def\datefinnish{%
  \def\today{\number\day.~\ifcase\month\or
    tammikuuta\or helmikuuta\or maaliskuuta\or huhtikuuta\or
    toukokuuta\or kes\"akuuta\or hein\"akuuta\or elokuuta\or
    syyskuuta\or lokakuuta\or marraskuuta\or joulukuuta\fi
    \space\number\year}}
```

`\extrasfinnish` `\noextrasfinnish` Finnish has many long words (some of them compound, some not). For this reason hyphenation is very often the only solution in line breaking. For this reason the values of `\hyphenpenalty`, `\exhyphenpenalty` and `\doublehyphendemerits` should be decreased. (In one of the manuals of style Matti Rintala noticed a paragraph with ten lines, eight of which ended in a hyphen!)

Matti Rintala noticed that with these changes TEX handles Finnish very well, although sometimes the values of `\tolerance` and `\emergencystretch` must be increased. However, I don't think changing these values in `finnish.ldf` is appropriate, as the looseness of the font (and the line width) affect the correct choice of these parameters.

```
\addto\extrasfinnish{%
  \babel@savevariable\hyphenpenalty\hyphenpenalty=30%
  \babel@savevariable\exhyphenpenalty\exhyphenpenalty=30%
  \babel@savevariable\doublehyphendemerits\doublehyphendemerits=5000%
  \babel@savevariable\finalhyphendemerits\finalhyphendemerits=5000%
  }
\addto\noextrasfinnish{}
```

Another thing `\extrasfinnish` needs to do is to ensure that `\frenchspacing` is in effect. If this is not the case the execution of `\noextrasfinnish` will switch it of again.

```
\addto\extrasfinnish{\bbl@frenchspacing}
\addto\noextrasfinnish{\bbl@nonfrenchspacing}
```

For Finnish the `"` character is made active. This is done once, later on its definition may vary. Other languages in the same document may also use the `"` character for shorthands; we specify that the finnish group of shorthands should be used.

```
\initiate@active@char{"}
\addto\extrasfinnish{\languageshorthands{finnish}}
```

Don't forget to turn the shorthands off again.

```
\addto\extrasfinnish{\bbl@activate{"}}
\addto\noextrasfinnish{\bbl@deactivate{"}}
```

The 'umlaut' character should be positioned lower on *all* vowels in Finnish texts.

```
\addto\extrasfinnish{\umlautlow\umlautelow}
\addto\noextrasfinnish{\umlauthigh}
```

First we define access to the low opening double quote and guillemets for quotations,

```
\declare@shorthand{finnish}{"`}{%
  \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
\declare@shorthand{finnish}{"'}{%
  \textormath{\textquotedblright}{\mbox{\textquotedblright}}}
\declare@shorthand{finnish}{"<}{%
  \textormath{\guillemotleft}{\mbox{\guillemotleft}}}
\declare@shorthand{finnish}{">}{%
  \textormath{\guillemotright}{\mbox{\guillemotright}}}
```

then we define two shorthands to be able to specify hyphenation breakpoints that behave a little different from `\-`.

```
\declare@shorthand{finnish}{"-}{\nobreak-\bbl@allowhyphens}
\declare@shorthand{finnish}{""}{\hskip\z@skip}
\declare@shorthand{finnish}{"=}{\hbox{-}\bbl@allowhyphens}
```

And we want to have a shorthand for disabling a ligature.

```
\declare@shorthand{finnish}{"|}{%
  \textormath{\discretionary{-}{}{\kern.03em}}{}}
```

`\-` All that is left now is the redefinition of `\-`. The new version of `\-` should indicate an extra hyphenation position, while allowing other hyphenation positions to be generated automatically. The standard behaviour of TEX in this respect is very unfortunate for languages such as Dutch, Finnish and German, where long compound words are quite normal and all one needs is a means to indicate an extra hyphenation position on top of the ones that TEX can generate from the hyphenation patterns.

```
\addto\extrasfinnish{\babel@save\-}
\addto\extrasfinnish{\def\-{\bbl@allowhyphens
```

```
                                 \discretionary{-}{}{}\bbl@allowhyphens}}
```

`\finishhyphenmins` The finnish hyphenation patterns can be used with `\lefthyphenmin` set to 2 and `\righthyphenmin` set to 2.

```
\providehyphenmins{\CurrentOption}{\tw@\tw@}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{finnish}
⟨/code⟩
```

## 46 The Hungarian language

The file option `magyar.dtx` defines all the language definition macros for the Hungarian language.

The babel support for the Hungarian language until file version 1.3i was essentially changing the English document elements to Hungarian ones, but because of the differences between these too languages this was actually unusable ('Part I' was transferred to 'Rész I' which is not usable instead of 'I. rész'). To enhance the typesetting facilities for Hungarian the following should be considered:

- In Hungarian documents there is a period after the part, section, subsection etc. numbers.
- In the part, chapter, appendix name the number (or letter) goes before the name, so 'Part I' translates to 'I. rész'.
- The same is true with captions ('Table 2.1' goes to '2.1. táblázat').
- There is a period after the caption name instead of a colon. ('Table 2.1:' goes to '2.1. táblázat.')
- There is a period at the end of the title in a run-in head (when `afterskip<0` in `\@startsection`).
- Special hyphenation rules must be applied for the so-called long double consonants (ccs, ssz,...).
- The opening quotation mark is like the German one (the closing is the same as in English).
- In Hungarian figure, table, etc. referencing a definite article is also incorporated. The Hungarian definite articles behave like the English indefinite ones ('a/an'). 'a' is used for words beginning with a consonant and 'az' goes for a vowel. Since some numbers begin with a vowel some others with a consonant some commands should be provided for automatic definite article generation.

Until file version 1.3i[53] the special typesetting rules of the Hungarian language mentioned above were not taken into consideration. This version (v1.4j)[54] enables babel to typeset 'good-looking' Hungarian texts.

`\ontoday` The `\ontoday` command works like `\today` but produces a slightly different date format used in expressions such as 'on February 10th'.

`\Az` The commands `\Az#1` and `\az#1` write the correct definite article for the argument and the argument itself (separated with a `~`). The star-forms (`\Az*` and `\az*`) produce the article only.

`\Azr` `\Azr#1` and `\azr#1` treat the argument as a label so expand it then write the

[53]That file was last revised on 1996/12/23 with a contribution by the next authors: Attila Koppányi (`attila@cernvm.cern.ch`), Árpád Bíró (`JZP1104@HUSZEG11.bitnet`), István Hamecz (`hami@ursus.bke.hu`) and Dezső Horváth (`horvath@pisa.infn.it`).

[54]It was written by József Bérces (`jozsi@docs4.mht.bme.hu`) with some help from Ferenc Wettl (`wettl@math.bme.hu`) and an idea from David Carlisle (`david@dcarlisle.demon.co.uk`).

| shortcut | explanation | example |
|---|---|---|
| `‘‘` | same as `\glqq` in babel, or `\quotedblbase` in T1 (opening quotation mark, like „) | `‘‘id\’ezet’’`⟶„idézet” |
| `‘c`, `‘C` | ccs is hyphenated as cs-cs | `lo‘ccsan`⟶locs-csan |
| `‘d`, `‘D` | ddz is hyphenated as dz-dz | `e‘ddz\"unk`⟶edz-dzünk |
| `‘g`, `‘G` | ggy is hyphenated as gy-gy | `po‘ggy\’asz`⟶pogy-gyász |
| `‘l`, `‘L` | lly is hyphenated as ly-ly | `Kod\’a‘llyal`⟶Kodály-lyal |
| `‘n`, `‘N` | nny is hyphenated as ny-ny | `me‘nnyei`⟶meny-nyei |
| `‘s`, `‘S` | ssz is hyphenated as sz-sz | `vi‘ssza`⟶visz-sza |
| `‘t`, `‘T` | tty is hyphenated as ty-ty | `po‘ttyan`⟶poty-tyan |
| `‘z`, `‘Z` | zzs is hyphenated as zs-zs | `ri‘zzsel`⟶rizs-zsel |

Table 20: The shortcuts defined in `magyar.ldf`

definite article for `\r@#1`, a non-breakable space then the label expansion. The star-forms do not print the label expansion. `\Azr(#1` and `\azr(#1` are used for equation referencing with the syntax `\azr(`*label*`)`.

`\Aref` There are two aliases `\Aref` and `\aref` for `\Azr` and `\azr`, respectively. During the preparation of a document it is not known in general, if the code ‘`a~\ref{`*label*`}`’ or the code ‘`az~\ref{`*label*`}`’ is the grammatically correct one. Writing ‘`\aref{`*label*`}`’ instead of the previous ones solves the problem.

`\Azp` `\Azp#1` and `\azp#1` also treat the argument as a label but use the label’s page for definite article determination. There are star-forms giving only the definite article without the page number.

`\Apageref` There are aliases `\Apageref` and `\apageref` for `\Azp` and `\azp`, respectively. The code `\apageref{`*label*`}` is equivalent either to `a~\pageref{`*label*`}` or to `az~\pageref{`*label*`}`.

`\Azc` `\Azc` and `\azc` work like the `\cite` command but (of course) they insert the definite article. There can be several comma separated cite labels and in that case the definite article is given for the first one. They accept `\cite`’s optional argument. There are star-forms giving the definite article only.

`\Acite` There are aliases `\Acite` and `\acite` for `\Azc` and `\azc`, respectively.

For this language the character `‘` is made active. Table 20 shows the shortcuts. The main reason for the activation of the `‘` character is to handle the special hyphenation of the long double consonants.

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
46.1 ⟨*code⟩
46.2 \LdfInit{magyar}{caption\CurrentOption}
```

When this file is read as an option, i.e. by the `\usepackage` command, `magyar` will be an ‘unknown’ language in which case we have to make it known. So we check for the existence of `\l@magyar` or `\l@hungarian` to see whether we have to do something here.

```
\ifx\l@magyar\@undefined
  \ifx\l@hungarian\@undefined
    \@nopatterns{Magyar}
    \adddialect\l@magyar0
  \else
    \let\l@magyar\l@hungarian
  \fi
\fi
```

The statement above makes sure that `\l@magyar` is always defined; if `\l@hungarian` is still undefined we make it equal to `\l@magyar`.

```
\ifx\l@hungarian\@undefined
  \let\l@hungarian\l@magyar
\fi
```

The next step consists of defining commands to switch to (and from) the Hungarian language.

`\captionsmagyar` The macro `\captionsmagyar` defines all strings used in the four standard document classes provided with LaTeX.

```
\@namedef{captions\CurrentOption}{%
  \def\prefacename{El\H osz\'o}%
```

For the list of references at the end of an article we have a choice between two words, ‘Referenciák’ (a Hungarian version of the English word) and ‘Hivatkozások’. The latter seems to be in more widespread use.

```
  \def\refname{Hivatkoz\'asok}%
```

If you have a document with a summary instead of an abstract you might want to replace the word ‘Kivonat’ with ‘Összefoglaló’.

```
  \def\abstractname{Kivonat}%
```

The Hungarian version of ‘Bibliography’ is ‘Bibliográfia’, but a more natural word to use is ‘Irodalomjegyzék’.

```
  \def\bibname{Irodalomjegyz\'ek}%
  \def\chaptername{fejezet}%
  \def\appendixname{F\"uggel\'ek}%
  \def\contentsname{Tartalomjegyz\'ek}%
  \def\listfigurename{\'Abr\'ak jegyz\'eke}%
  \def\listtablename{T\'abl\'azatok jegyz\'eke}%
  \def\indexname{T\'argymutat\'o}%
  \def\figurename{\'abra}%
  \def\tablename{t\'abl\'azat}%
  \def\partname{r\'esz}%
  \def\enclname{Mell\'eklet}%
  \def\ccname{K\"orlev\'el--c\'\i mzettek}%
  \def\headtoname{C\'\i mzett}%
  \def\pagename{oldal}%
  \def\seename{l\'asd}%
  \def\alsoname{l\'asd m\'eg}%
```

Besides the Hungarian word for Proof, 'Bizonyítás' we can also name Corollary (Következmény), Theorem (Tétel) and Lemma (Lemma).

```
  \def\proofname{Bizony\'\i t\'as}%
  \def\glossaryname{Sz\'ojegyz\'ek}%
  }%
```

\datemagyar The macro `\datemagyar` redefines the command `\today` to produce Hungarian dates.

```
\@namedef{date\CurrentOption}{%
  \def\today{%
    \number\year.\nobreakspace\ifcase\month\or
    janu\'ar\or febru\'ar\or m\'arcius\or
    \'aprilis\or m\'ajus\or j\'unius\or
    j\'ulius\or augusztus\or szeptember\or
    okt\'ober\or november\or december\fi
    \space\number\day.}}
```

\ondatemagyar The macro `\ondatemagyar` produces Hungarian dates which have the meaning '*on this day*'. It does not redefine the command `\today`.

```
\@namedef{ondate\CurrentOption}{%
  \number\year.\nobreakspace\ifcase\month\or
  janu\'ar\or febru\'ar\or m\'arcius\or
  \'aprilis\or m\'ajus\or j\'unius\or
  j\'ulius\or augusztus\or szeptember\or
  okt\'ober\or november\or december\fi
    \space\ifcase\day\or
    1-j\'en\or  2-\'an\or  3-\'an\or  4-\'en\or  5-\'en\or
    6-\'an\or  7-\'en\or  8-\'an\or  9-\'en\or 10-\'en\or
   11-\'en\or 12-\'en\or 13-\'an\or 14-\'en\or 15-\'en\or
   16-\'an\or 17-\'en\or 18-\'an\or 19-\'en\or 20-\'an\or
   21-\'en\or 22-\'en\or 23-\'an\or 24-\'en\or 25-\'en\or
   26-\'an\or 27-\'en\or 28-\'an\or 29-\'en\or 30-\'an\or
   31-\'en\fi}
```

\extrasmagyar \noextrasmagyar The macro `\extrasmagyar` will perform all the extra definitions needed for the Hungarian language. The macro `\noextrasmagyar` is used to cancel the actions of `\extrasmagyar`.

```
 \@namedef{extras\CurrentOption}{%
   \expandafter\let\expandafter\ontoday
     \csname ondate\CurrentOption\endcsname}
\@namedef{noextras\CurrentOption}{\let\ontoday\@undefined}
```

Now we redefine some commands included into `latex.ltx`. The original form of a command is always saved with `\babel@save` and the changes are added to `\extrasmagyar`. This ensures that the Hungarian version of a macro is alive *only* if the Hungarian language is active.

\fnum@figure \fnum@table In figure and table captions the order of the figure/table number and \figurename /\tablename must be changed. To achieve this \fnum@figure and \fnum@table are redefined and added to \extrasmagyar.

```
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \babel@save\fnum@figure
  \def\fnum@figure{\thefigure.\nobreakspace\figurename}}
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \babel@save\fnum@table
  \def\fnum@table{\thetable.\nobreakspace\tablename}}
```

\@makecaption The colon in a figure/table caption must be replaced by a dot by redefining \@makecaption.

```
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \babel@save\@makecaption
  \def\@makecaption#1#2{%
    \vskip\abovecaptionskip
    \sbox\@tempboxa{#1. #2}%
    \ifdim \wd\@tempboxa >\hsize
      {#1. #2\csname par\endcsname}
    \else
      \global \@minipagefalse
      \hb@xt@\hsize{\hfil\box\@tempboxa\hfil}%
    \fi
    \vskip\belowcaptionskip}}
```

\@caption There should be a dot after the figure/table number in lof/lot, so \@caption is redefined.

```
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \babel@save\@caption
  \long\def\@caption#1[#2]#3{%
    \csname par\endcsname
    \addcontentsline{\csname ext@#1\endcsname}{#1}%
      {\protect\numberline{\csname the#1\endcsname.}{\ignorespaces #2}}%
    \begingroup
      \@parboxrestore
      \if@minipage
        \@setminipage
      \fi
      \normalsize
      \@makecaption{\csname fnum@#1\endcsname}%
          {\ignorespaces #3}\csname par\endcsname
    \endgroup}}
```

\@seccntformat In order to have a dot after the section number \@seccntformat is redefined.

```
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \babel@save\@seccntformat
  \def\@seccntformat#1{\csname the#1\endcsname.\quad}}
```

`\@sect` Alas, `\@sect` must also be redefined to have that dot in toc too. On the other hand, we include a dot after a run-in head.

```
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \babel@save\@sect
  \def\@sect#1#2#3#4#5#6[#7]#8{%
    \ifnum #2>\c@secnumdepth
      \let\@svsec\@empty
    \else
      \refstepcounter{#1}%
      \protected@edef\@svsec{\@seccntformat{#1}\relax}%
    \fi
    \@tempskipa #5\relax
    \ifdim \@tempskipa>\z@
      \begingroup
        #6{%
          \@hangfrom{\hskip #3\relax\@svsec}%
            \interlinepenalty \@M #8\@@par}%
      \endgroup
      \csname #1mark\endcsname{#7}%
      \addcontentsline{toc}{#1}{%
        \ifnum #2>\c@secnumdepth \else
          \protect\numberline{\csname the#1\endcsname.}%
        \fi
        #7}%
    \else
      \def\@svsechd{%
        #6{\hskip #3\relax
        \@svsec #8.}%
        \csname #1mark\endcsname{#7}%
        \addcontentsline{toc}{#1}{%
          \ifnum #2>\c@secnumdepth \else
            \protect\numberline{\csname the#1\endcsname.}%
          \fi
          #7}}%
    \fi
    \@xsect{#5}}}
```

`\@ssect` In order to have that dot after a run-in head when the star form of the sectioning commands is used, we have to redefine `\@ssect`.

```
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \babel@save\@ssect
  \def\@ssect#1#2#3#4#5{%
    \@tempskipa #3\relax
    \ifdim \@tempskipa>\z@
      \begingroup
        #4{%
          \@hangfrom{\hskip #1}%
            \interlinepenalty \@M #5\@@par}%
      \endgroup
    \else
      \def\@svsechd{#4{\hskip #1\relax #5.}}%
    \fi
    \@xsect{#3}}}
```

\@begintheorem \@opargbegintheorem Order changing and dot insertion in theorem by redefining \@begintheorem and \@opargbegintheorem.

```
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \babel@save\@begintheorem
  \def\@begintheorem#1#2{\trivlist
    \item[\hskip \labelsep{\bfseries #2.\ #1.}]\itshape}%
  \babel@save\@opargbegintheorem
  \def\@opargbegintheorem#1#2#3{\trivlist
    \item[\hskip \labelsep{\bfseries #2.\ #1\ (#3).}]\itshape}}
```

The next step is to redefine some macros included into the class files. It is determined which class file is loaded then the original form of the macro is saved and the changes are added to \extrasmagyar.

First we check if the book.cls is loaded.

```
\@ifclassloaded{book}{%
```

\ps@headings The look of the headings is changed: we have to insert some dots and change the order of chapter number and \chaptername.

```
  \expandafter\addto\csname extras\CurrentOption\endcsname{%
    \babel@save\ps@headings}
  \expandafter\addto\csname extras\CurrentOption\endcsname{%
    \if@twoside
      \def\ps@headings{%
          \let\@oddfoot\@empty\let\@evenfoot\@empty
          \def\@evenhead{\thepage\hfil\slshape\leftmark}%
          \def\@oddhead{{\slshape\rightmark}\hfil\thepage}%
          \let\@mkboth\markboth
        \def\chaptermark##1{%
          \markboth {\MakeUppercase{%
            \ifnum \c@secnumdepth >\m@ne
              \if@mainmatter
                \thechapter. \@chapapp. \ %
              \fi
            \fi
            ##1}}{}}%
        \def\sectionmark##1{%
          \markright {\MakeUppercase{%
            \ifnum \c@secnumdepth >\z@
              \thesection. \ %
            \fi
            ##1}}}}%
    \else
      \def\ps@headings{%
        \let\@oddfoot\@empty
      \def\@oddhead{{\slshape\rightmark}\hfil\thepage}%
      \let\@mkboth\markboth
      \def\chaptermark##1{%
        \markright {\MakeUppercase{%
          \ifnum \c@secnumdepth >\m@ne
            \if@mainmatter
              \thechapter. \@chapapp. \ %
            \fi
          \fi
          ##1}}}}%
  \fi}
```

`\@part` At the beginning of a part we need eg. 'I. rész' instead of 'Part I' (in toc too). To achieve this `\@part` is redefined.

```
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \babel@save\@part
  \def\@part[#1]#2{%
      \ifnum \c@secnumdepth >-2\relax
        \refstepcounter{part}%
        \addcontentsline{toc}{part}{\thepart.\hspace{1em}#1}%
      \else
        \addcontentsline{toc}{part}{#1}%
      \fi
      \markboth{}{}%
      {\centering
       \interlinepenalty \@M
       \normalfont
       \ifnum \c@secnumdepth >-2\relax
         \huge\bfseries \thepart.\nobreakspace\partname
         \csname par\endcsname
         \vskip 20\p@
       \fi
       \Huge \bfseries #2\csname par\endcsname}%
      \@endpart}}
```

`\@chapter` The same changes are made to chapter. First the screen typeout and the toc are changed by redefining `\@chapter`.

```
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \babel@save\@chapter
  \def\@chapter[#1]#2{\ifnum \c@secnumdepth >\m@ne
                        \if@mainmatter
                          \refstepcounter{chapter}%
                          \typeout{\thechapter.\space\@chapapp.}%
                          \addcontentsline{toc}{chapter}%
                                    {\protect\numberline{\thechapter.}#1}%
                        \else
                          \addcontentsline{toc}{chapter}{#1}%
                        \fi
                      \else
```

```
 \addcontentsline{toc}{chapter}{#1}%
\fi
\chaptermark{#1}%
\addtocontents{lof}{\protect\addvspace{10\p@}}%
\addtocontents{lot}{\protect\addvspace{10\p@}}%
\if@twocolumn
 \@topnewpage[\@makechapterhead{#2}]%
\else
 \@makechapterhead{#2}%
 \@afterheading
\fi}}
```

`\@makechapterhead` Then the look of the chapter-start is modified by redefining `\@makechapterhead`.

```
 \expandafter\addto\csname extras\CurrentOption\endcsname{%
  \babel@save\@makechapterhead
  \def\@makechapterhead#1{%
   \vspace*{50\p@}%
   {\parindent \z@ \raggedright \normalfont
    \ifnum \c@secnumdepth >\m@ne
     \if@mainmatter
      \huge\bfseries \thechapter.\nobreakspace\@chapapp{}
      \csname par\endcsname\nobreak
      \vskip 20\p@
     \fi
    \fi
    \interlinepenalty\@M
    \Huge \bfseries #1\csname par\endcsname\nobreak
    \vskip 40\p@
   }}}%
```

This the end of the book class modification.

```
}{}
```

Now we check if `report.cls` is loaded.

```
\@ifclassloaded{report}{%
```

`\ps@headings` First the headings are modified just in case of the book class.

```
 \expandafter\addto\csname extras\CurrentOption\endcsname{%
  \babel@save\ps@headings}
 \expandafter\addto\csname extras\CurrentOption\endcsname{%
  \if@twoside
   \def\ps@headings{%
     \let\@oddfoot\@empty\let\@evenfoot\@empty
     \def\@evenhead{\thepage\hfil\slshape\leftmark}%
     \def\@oddhead{{\slshape\rightmark}\hfil\thepage}%
     \let\@mkboth\markboth
    \def\chaptermark##1{%
     \markboth {\MakeUppercase{%
      \ifnum \c@secnumdepth >\m@ne
        \thechapter. \@chapapp. \ %
```

```
            \fi
            ##1}}{}}%
        \def\sectionmark##1{%
          \markright {\MakeUppercase{%
            \ifnum \c@secnumdepth >\z@
              \thesection. \ %
            \fi
            ##1}}}}%
    \else
      \def\ps@headings{%
        \let\@oddfoot\@empty
        \def\@oddhead{{\slshape\rightmark}\hfil\thepage}%
        \let\@mkboth\markboth
        \def\chaptermark##1{%
          \markright {\MakeUppercase{%
            \ifnum \c@secnumdepth >\m@ne
                \thechapter. \@chapapp. \ %
            \fi
            ##1}}}}%
    \fi}
```

`\@chapter` Chapter-start modification with `\@chapter`

```
  \expandafter\addto\csname extras\CurrentOption\endcsname{%
    \babel@save\@chapter
    \def\@chapter[#1]#2{\ifnum \c@secnumdepth >\m@ne
                             \refstepcounter{chapter}%
                             \typeout{\thechapter.\space\@chapapp.}%
                             \addcontentsline{toc}{chapter}%
                                       {\protect\numberline{\thechapter.}#1}%
                        \else
                          \addcontentsline{toc}{chapter}{#1}%
                        \fi
                        \chaptermark{#1}%
                        \addtocontents{lof}{\protect\addvspace{10\p@}}%
                        \addtocontents{lot}{\protect\addvspace{10\p@}}%
                        \if@twocolumn
                          \@topnewpage[\@makechapterhead{#2}]%
                        \else
                          \@makechapterhead{#2}%
                          \@afterheading
                        \fi}}
```

`\@makechapterhead` and `\@makechapterhead`.

```
  \expandafter\addto\csname extras\CurrentOption\endcsname{%
    \babel@save\@makechapterhead
    \def\@makechapterhead#1{%
      \vspace*{50\p@}%
      {\parindent \z@ \raggedright \normalfont
        \ifnum \c@secnumdepth >\m@ne
            \huge\bfseries \thechapter.\nobreakspace\@chapapp{}
            \csname par\endcsname\nobreak
            \vskip 20\p@
        \fi
        \interlinepenalty\@M
        \Huge \bfseries #1\csname par\endcsname\nobreak
        \vskip 40\p@
      }}}%
```

End of report class modification.

```
}{}
```

Checking if `article.cls` is loaded.

```
\@ifclassloaded{article}{%
```

`\ps@headings` Changing headings by redefining `\ps@headings`.

```
  \expandafter\addto\csname extras\CurrentOption\endcsname{%
    \babel@save\ps@headings}
  \expandafter\addto\csname extras\CurrentOption\endcsname{%
    \if@twoside
      \def\ps@headings{%
          \let\@oddfoot\@empty\let\@evenfoot\@empty
          \def\@evenhead{\thepage\hfil\slshape\leftmark}%
          \def\@oddhead{{\slshape\rightmark}\hfil\thepage}%
          \let\@mkboth\markboth
        \def\sectionmark##1{%
          \markboth {\MakeUppercase{%
            \ifnum \c@secnumdepth >\z@
              \thesection.\quad
            \fi
            ##1}}{}}%
        \def\subsectionmark##1{%
          \markright {%
            \ifnum \c@secnumdepth >\@ne
              \thesubsection.\quad
            \fi
            ##1}}}%
    \else
      \def\ps@headings{%
        \let\@oddfoot\@empty
        \def\@oddhead{{\slshape\rightmark}\hfil\thepage}%
        \let\@mkboth\markboth
        \def\sectionmark##1{%
          \markright {\MakeUppercase{%
            \ifnum \c@secnumdepth >\m@ne
              \thesection.\quad
            \fi
            ##1}}}}%
    \fi}%
```

No more necessary changes specific to the article class.

```
}{}
```

And now this is the turn of `letter.cls`.

```
\@ifclassloaded{letter}{%
```

`\ps@headings` In the headings the page number must be followed by a dot and then `\pagename`.

```
  \expandafter\addto\csname extras\CurrentOption\endcsname{%
    \babel@save\ps@headings}
  \expandafter\addto\csname extras\CurrentOption\endcsname{%
    \if@twoside
      \def\ps@headings{%
          \let\@oddfoot\@empty\let\@evenfoot\@empty
          \def\@oddhead{\slshape\headtoname{:} \ignorespaces\toname
                        \hfil \@date
                        \hfil \thepage.\nobreakspace\pagename}%
          \let\@evenhead\@oddhead}
    \else
      \def\ps@headings{%
          \let\@oddfoot\@empty
          \def\@oddhead{\slshape\headtoname{:} \ignorespaces\toname
                        \hfil \@date
                        \hfil \thepage.\nobreakspace\pagename}}
    \fi}%
```

End of letter class.

```
}{}
```

After making the changes to the LaTeX macros we define some new ones to handle the problem with definite articles.

`\az` `\az` is a user-level command which decides if the next character is a star. `\@az` is called for `\az*` and `\az@` for `\az`.

```
\def\az{a\@ifstar{\@az}{\az@}}
```

`\Az` `\Az` is used at the beginning of a sentence. Otherwise it behaves the same as `\az`.

```
\def\Az{A\@ifstar{\@az}{\az@}}
```

`\az@` `\az@` is called if there is no star after `\az` or `\Az`. It calls `\@az` and writes `#1` separating with a non-breakable space.

```
\def\az@#1{\@az{#1}\nobreakspace#1}
```

`\@az` This macro calls `\hun@tempadef` to remove the accents from the argument then calls `\@@az` that determines if a ‘z’ should be written after a/A (written by `\az`/`\Az`).

```
\def\@az#1{%
  \hun@tempadef{relax}{relax}{#1}%
  \edef\@tempb{\noexpand\@@az\@tempa\hbox!}%
  \@tempb}
```

`\hun@tempadef` The macro `\hun@tempadef` has three tasks:

- Accent removal. Accented letters confuse `\@@az`, the main definite article determinator macro, so they must be changed to their non-accented counterparts. Special letters must also be changed, eg. œ → o.
- Labels must be expanded.
- To handle Roman numerals correctly, commands starting with `\hun@` are defined for labels containing Roman numbers with the Roman numerals replaced by their Arabic representation. This macro can check if there is a `\hun@` command.

There are three arguments:

1. The primary command that should be expanded if it exists. This is usually the `\hun@` command for a label.
2. The secondary command which is used if the first one is `\relax`. This is usually the original LaTeX command for a label.
3. This is used if the first two is `\relax`. For this one no expansion is carried out but the accents are still removed and special letters are changed.

```
\def\hun@tempadef#1#2#3{%
  \begingroup
    \def\@safe@activesfalse{}%
    \def\setbox ##1{}% to get rid of accents and special letters
    \def\hbox ##1{}%
    \def\accent ##1 ##2{##2}%
    \def\add@accent ##1##2{##2}%
    \def\@text@composite@x ##1##2{##2}%
    \def\i{i}\def\j{j}%
    \def\ae{a}\def\AE{A}\def\oe{o}\def\OE{O}%
    \def\ss{s}\def\L{L}%
    \def\d{}\def\b{}\def\c{}\def\t{}%
    \expandafter\ifx\csname #1\endcsname\relax
      \expandafter\ifx\csname #2\endcsname\relax
        \xdef\@tempa{#3}%
      \else
        \xdef\@tempa{\csname #2\endcsname}%
      \fi
    \else
      \xdef\@tempa{\csname #1\endcsname}%
    \fi
  \endgroup}
```

The following macros are used to determine the definite article for a label's expansion.

`\aref` `\aref` is an alias for `\azr`.

```
\def\aref{\azr}
```

\Aref \Aref is an alias for \Azr.

```
\def\Aref{\Azr}
```

\azr \azr calls \@azr if the next character is a star, otherwise it calls \azr@.

```
\def\azr{a\@ifstar{\@azr}{\azr@}}
```

\Azr \Azr is the same as \azr except that it writes ‘A’ instead of ‘a’.

```
\def\Azr{A\@ifstar{\@azr}{\azr@}}
```

\azr@ \azr@ decides if the next character is ( and in that case it calls \azr@@@ which writes an extra ( for equation referencing. Otherwise \azr@@ is called.

```
\def\azr@{\@ifnextchar ({\azr@@@}{\azr@@}}
```

\azr@@ Calls \@azr then writes the label’s expansion preceded by a non-breakable space.

```
\def\azr@@#1{\@azr{#1}\nobreakspace\ref{#1}}
```

\azr@@@ Same as \azr@@ but inserts a ( between the non-breakable space and the label expansion.

```
\def\azr@@@(#1{\@azr{#1}\nobreakspace(\ref{#1}}
```

\@azr Calls \hun@tempadef to choose between the label’s \hun@ or original LaTeX command and to expand it with accent removal and special letter substitution. Then calls \@@az, the core macro of definite article handling.

```
\def\@azr#1{%
  \hun@tempadef{hun@r@#1}{r@#1}{}%
  \ifx\@tempa\empty
  \else
    \edef\@tempb{\noexpand\@@az\expandafter\@firstoftwo\@tempa\hbox!}%
    \@tempb
  \fi
}
```

The following commands are used to generate the definite article for the page number of a label.

\apageref \apageref is an alias for \azp.

```
\def\apageref{\azp}
```

\Apageref \Apageref is an alias for \Azp.

```
\def\Apageref{\Azp}
```

\azp Checks if the next character is * and calls \@azp or \azp@.

```
\def\azp{a\@ifstar{\@azp}{\azp@}}
```

\Azp Same as \azp except that it writes ‘A’ instead of ‘a’.

```
\def\Azp{A\@ifstar{\@azp}{\azp@}}
```

`\azp@` Calls `\@azp` then writes the label's page preceded by a non-breakable space.

```
\def\azp@#1{\@azp{#1}\nobreakspace\pageref{#1}}
```

`\@azp` Calls `\hun@tempadef` then takes the label's page and passes it to `\@@az`.

```
\def\@azp#1{%
  \hun@tempadef{hun@r@#1}{r@#1}{}%
  \ifx\@tempa\empty
  \else
    \edef\@tempb{\noexpand\@@az\expandafter\@secondoftwo\@tempa\hbox!}%
    \@tempb
  \fi
}
```

The following macros are used to give the definite article to citations.

`\acite` This is an alias for `\azc`.

```
\def\acite{\azc}
```

`\Acite` This is an alias for `\Azc`.

```
\def\Acite{\Azc}
```

`\azc` Checks if the next character is a star and calls `\@azc` or `\azc@`.

```
\def\azc{a\@ifstar{\@azc}{\azc@}}
```

`\Azc` Same as `\azc` but used at the beginning of sentences.

```
\def\Azc{A\@ifstar{\@azc}{\azc@}}
```

`\azc@` If there is no star we accept an optional argument, just like the `\cite` command.

```
\def\azc@{\@ifnextchar [{\azc@@}{\azc@@[]}}
```

`\azc@@` First calls `\@azc` then `\cite`.

```
\def\azc@@[#1]#2{%
  \@azc{#2}\nobreakspace\def\@tempa{#1}%
    \ifx\@tempa\@empty\cite{#2}\else\cite[#1]{#2}\fi}
```

`\@azc` This is an auxiliary macro to get the first cite label from a comma-separated list.

```
\def\@azc#1{\@@azc#1,\hbox!}
```

`\@@azc` This one uses only the first argument, that is the first element of the comma-separated list of cite labels. Calls `\hun@tempadef` to expand the cite label with accent removal and special letter replacement. Then `\@@az`, the core macro, is called.

```
\def\@@azc#1,#2\hbox#3!{%
  \hun@tempadef{hun@b@#1}{b@#1}{}%
  \ifx\@tempa\empty
  \else
    \edef\@tempb{\noexpand\@@az\@tempa\hbox!}%
    \@tempb
  \fi}
```

`\hun@number@lehgth` This macro is used to count the number of digits in its argument until a non-digit character is found or the end of the argument is reached. It must be called as `\hun@number@lehgth`*arg*`\hbox\hbox!` and `\count@` must be zeroed. It is called by `\@@az`.

```
\def\hun@number@lehgth#1#2\hbox#3!{%
  \ifcat\noexpand#11%
    \ifnum\expandafter`\csname#1\endcsname>47
      \ifnum\expandafter`\csname#1\endcsname<58
        \advance\count@ by \@ne
        \hun@number@lehgth#2\hbox\hbox!\fi\fi\fi}
```

`\hun@alph@lehgth` This is used to count the number of letters until a non-letter is found or the end of the argument is reached. It must be called as `\hun@alph@lehgth`*arg*`\hbox\hbox!` and `\count@` must be set to zero. It is called by `\@@az@string`.

```
\def\hun@alph@lehgth#1#2\hbox#3!{%
  \ifcat\noexpand#1A%
    \advance\count@ by \@ne
    \hun@alph@lehgth#2\hbox\hbox!\fi}
```

`\@@az@string` This macro is called by `\@@az` if the argument begins with a letter. The task of `\@@az@string` is to determine if the argument starts with a vowel and in that case `\let\@tempa\@tempb`. After checking if the first letter is A, E, I, O, or U, `\hun@alph@lehgth` is called to determine the length of the argument. If it gives 1 (that is the argument is a single-letter one or the second character is not letter) then the letters L, M, N, R, S, X, and Y are also considered as a vowel since their Hungarian pronounced name starts with a vowel.

```
\def\@@az@string#1#2{%
  \ifx#1A%
    \let\@tempa\@tempb
  \else\ifx#1E%
    \let\@tempa\@tempb
  \else\ifx#1I%
    \let\@tempa\@tempb
  \else\ifx#1O%
    \let\@tempa\@tempb
  \else\ifx#1U%
    \let\@tempa\@tempb
  \fi\fi\fi\fi\fi
  \ifx\@tempa\@tempb
  \else
    \count@\z@
    \hun@alph@lehgth#1#2\hbox\hbox!%
    \ifnum\count@=\@ne
      \ifx#1F%
        \let\@tempa\@tempb
      \else\ifx#1L%
        \let\@tempa\@tempb
      \else\ifx#1M%
```

```
        \let\@tempa\@tempb
      \else\ifx#1N%
        \let\@tempa\@tempb
      \else\ifx#1R%
        \let\@tempa\@tempb
      \else\ifx#1S%
        \let\@tempa\@tempb
      \else\ifx#1X%
        \let\@tempa\@tempb
      \else\ifx#1Y%
        \let\@tempa\@tempb
      \fi\fi\fi\fi\fi\fi\fi\fi
    \fi
  \fi}
```

`\@@az` This macro is the core of definite article handling. It determines if the argument needs 'az' or 'a' definite article by setting `\@tempa` to 'z' or `\@empty`. It sets `\@tempa` to 'z' if

- the first character of the argument is 5; or
- the first character of the argument is 1 and the *length of the number* (mod 3) = 1 (one–egy, thousand–ezer, million–egymillió,...); or
- the first character of the argument is a, A, e, E, i, I, o, O, u, or U; or
- the first character of the argument is l, L, m, M, n, N, r, R, s, S, x, X, y, or Y and the length of the argument is 1 or the second character is a non-letter.

At the end it calls `\@tempa`, that is, it either typesets a 'z' or nothing.

```
\def\@@az#1#2\hbox#3!{%
  \let\@tempa\@empty
  \def\@tempb{z}%
  \uppercase{%
    \ifx5#1%
      \let\@tempa\@tempb
    \else\ifx1#1%
      \count@\@ne
      \hun@number@lehgth#2\hbox\hbox!%
      \loop
      \ifnum\count@>\thr@@
        \advance\count@-\thr@@
      \repeat
      \ifnum\count@=\@ne
        \let\@tempa\@tempb
      \fi
    \else
      \@@az@string{#1}{#2}%
    \fi\fi
  }%
  \@tempa}
```

`\refstepcounter` `\refstepcounter` must be redefined in order to keep `\@currentlabel` unexpanded. This is necessary to enable the `\label` command to write a `\hunnewlabel` command to the aux file with the Roman numerals substituted by their Arabic representations. Of course, the original definition of `\refstepcounter` is saved and restored if the Hungarian language is switched off.

```
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \babel@save\refstepcounter
  \def\refstepcounter#1{\stepcounter{#1}%
    \def\@currentlabel{\csname p@#1\endcsname\csname the#1\endcsname}}%
}
```

`\label` `\label` is redefined to write another line into the aux file: `\hunnewlabel{ }{ }` where the Roman numerals are replaced their Arabic representations. The original definition of `\label` is saved into `\old@label` and it is also called by `\label`. On leaving the Hungarian typesetting mode `\label`'s original is restored since it is added to `\noextrasmagyar`.

```
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \let\old@label\label
  \def\label#1{\@bsphack
    \old@label{#1}%
    \begingroup
      \let\romannumeral\number
      \def\@roman##1{\number ##1}%
      \def\@Roman##1{\number ##1}%
      {\toks0={\noexpand\noexpand\noexpand\number}%
        \def\number##1{\the\toks0 ##1}\xdef\tempb@{\thepage}}%
      \edef\@tempa##1{\noexpand\protected@write\@auxout{}%
           {\noexpand\string\noexpand\hunnewlabel
           {##1}{{\@currentlabel}{\tempb@}}}}%
      \@tempa{#1}%
    \endgroup
  \@esphack}%
}
\expandafter\addto\csname noextras\CurrentOption\endcsname{%
  \let\label\old@label
}
```

`\hunnewlabel` Finally, `\hunnewlabel` is defined. It checks if the label's expansion (`#2`) differs from that one given in the `\newlabel` command. If yes (that is, the label contains some Roman numerals), it defines the macro `\hun@r@`*label*, otherwise it does nothing.

```
\def\hunnewlabel#1#2{%
  \def\@tempa{#2}%
  \expandafter\ifx\csname r@#1\endcsname\@tempa
    \relax% \message{No need for def: #1}%
  \else
    \global\expandafter\let\csname hun@r@#1\endcsname\@tempa%
  \fi
}
```

For Hungarian the ` character is made active.

```
\AtBeginDocument{%
  \if@filesw\immediate\write\@auxout{\catcode096=12}\fi}
\initiate@active@char{`}
\expandafter\addto\csname extras\CurrentOption\endcsname{%
  \languageshorthands{magyar}%
  \bbl@activate{`}}
\expandafter\addto\csname noextras\CurrentOption\endcsname{%
  \bbl@deactivate{`}}
```

The character sequence `` is declared as a shorthand in order to produce the opening quotation sign appropriate for Hungarian.

```
\declare@shorthand{magyar}{``}{\glqq}
```

In Hungarian there are some long double consonants which must be hyphenated specially. For all these long double consonants (except dzzs, that is extremely very-very rare) a shortcut is defined.

```
\declare@shorthand{magyar}{`c}{\textormath{\bbl@disc{c}{cs}}{c}}
\declare@shorthand{magyar}{`C}{\textormath{\bbl@disc{C}{CS}}{C}}
\declare@shorthand{magyar}{`d}{\textormath{\bbl@disc{d}{dz}}{d}}
\declare@shorthand{magyar}{`D}{\textormath{\bbl@disc{D}{DZ}}{D}}
\declare@shorthand{magyar}{`g}{\textormath{\bbl@disc{g}{gy}}{g}}
\declare@shorthand{magyar}{`G}{\textormath{\bbl@disc{G}{GY}}{G}}
\declare@shorthand{magyar}{`l}{\textormath{\bbl@disc{l}{ly}}{l}}
\declare@shorthand{magyar}{`L}{\textormath{\bbl@disc{L}{LY}}{L}}
\declare@shorthand{magyar}{`n}{\textormath{\bbl@disc{n}{ny}}{n}}
\declare@shorthand{magyar}{`N}{\textormath{\bbl@disc{N}{NY}}{N}}
\declare@shorthand{magyar}{`s}{\textormath{\bbl@disc{s}{sz}}{s}}
\declare@shorthand{magyar}{`S}{\textormath{\bbl@disc{S}{SZ}}{S}}
\declare@shorthand{magyar}{`t}{\textormath{\bbl@disc{t}{ty}}{t}}
\declare@shorthand{magyar}{`T}{\textormath{\bbl@disc{T}{TY}}{T}}
\declare@shorthand{magyar}{`z}{\textormath{\bbl@disc{z}{zs}}{z}}
\declare@shorthand{magyar}{`Z}{\textormath{\bbl@disc{Z}{ZS}}{Z}}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish\CurrentOption
⟨/code⟩
```

## 47 The Estonian language

The file `estonian.dtx`[55] defines the language definition macro's for the Estonian language.

This file was written as part of the TWGML project, and borrows heavily from the babel German and Spanish language files `germanb.ldf` and `spanish.ldf`.

Estonian has the same umlauts as German (ä, ö, ü), but in addition to this, we have also õ, and two recent characters š and ž, so we need at least two active characters. We shall use `"` and `~` to type Estonian accents on ASCII keyboards (in the 7-bit character world). Their use is given in table 21. These active accent

| | |
|---|---|
| `~o` | `\~o`, (and uppercase); |
| `"a` | `\"a`, (and uppercase); |
| `"o` | `\"o`, (and uppercase); |
| `"u` | `\"u`, (and uppercase); |
| `~s` | `\v s`, (and uppercase); |
| `~z` | `\v z`, (and uppercase); |
| `"\|` | disable ligature at this position; |
| `"-` | an explicit hyphen sign, allowing hyphenation in the rest of the word; |
| `\-` | like the old `\-`, but allowing hyphenation in the rest of the word; |
| ``"` `` | for Estonian low left double quotes (same as German); |
| `"'` | for Estonian right double quotes; |
| `"<` | for French left double quotes (also rather popular) |
| `">` | for French right double quotes. |

Table 21: The extra definitions made by `estonian.ldf`

characters behave according to their original definitions if not followed by one of the characters indicated in that table; the original quote character can be typed using the macro `\dq`.

We support also the T1 output encoding (and Cork-encoded text input). You can choose the T1 encoding by the command `\usepackage[T1]{fontenc}`. This package must be loaded before babel. As the standard Estonian hyphenation file `eehyph.tex` is in the Cork encoding, choosing this encoding will give you better hyphenation.

As mentioned in the Spanish style file, it may happen that some packages fail (usually in a `\message`). In this case you should change the order of the `\usepackage` declarations or the order of the style options in `\documentclass`.

[55]The file described in this section has version number v1.0h and was last revised on 2005/03/30. The original author is Enn Saar, (`saar@aai.ee`).

## 47.1 Implementation

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
⟨*code⟩
\LdfInit{estonian}\captionsestonian
```

If Estonian is not included in the format file (does not have hyphenation patterns), we shall use English hyphenation.

```
\ifx\l@estonian\@undefined
  \@nopatterns{Estonian}
  \adddialect\l@estonian0
\fi
```

Now come the commands to switch to (and from) Estonian.

`\captionsestonian` The macro `\captionsestonian` defines all strings used in the four standard documentclasses provided with LaTeX.

```
\addto\captionsestonian{%
  \def\prefacename{Sissejuhatus}%
  \def\refname{Viited}%
  \def\bibname{Kirjandus}%
  \def\appendixname{Lisa}%
  \def\contentsname{Sisukord}%
  \def\listfigurename{Joonised}%
  \def\listtablename{Tabelid}%
  \def\indexname{Indeks}%
  \def\figurename{Joonis}%
  \def\tablename{Tabel}%
  \def\partname{Osa}%
  \def\enclname{Lisa(d)}%
  \def\ccname{Koopia(d)}%
  \def\headtoname{}%
  \def\pagename{Lk.}%
  \def\seename{vt.}%
  \def\alsoname{vt. ka}%
  \def\proofname{Korrektuur}%
  \def\glossaryname{Glossary}% <-- Needs translation
  }
```

These captions contain accented characters.

```
\begingroup \catcode`\"\active
\def\x{\endgroup
\addto\captionsestonian{%
  \def\abstractname{Kokkuv~ote}%
  \def\chaptername{Peat"ukk}}}
\x
```

`\dateestonian` The macro `\dateestonian` redefines the command `\today` to produce Estonian dates.

```
\begingroup \catcode`\"\active
\def\x{\endgroup
  \def\month@estonian{\ifcase\month\or
    jaanuar\or veebruar\or m"arts\or aprill\or mai\or juuni\or
    juuli\or august\or september\or oktoober\or november\or
    detsember\fi}}
\x
\def\dateestonian{%
  \def\today{\number\day.\space\month@estonian
    \space\number\year.\space a.}}
```

\extrasestonian
\noextrasestonian

The macro `\extrasestonian` will perform all the extra definitions needed for Estonian. The macro `\noextrasestonian` is used to cancel the actions of `\extrasestonian`. For Estonian, `"` is made active and has to be treated as 'special' (`~` is active already).

```
\initiate@active@char{"}
\initiate@active@char{~}
\addto\extrasestonian{\languageshorthands{estonian}}
\addto\extrasestonian{\bbl@activate{"}\bbl@activate{~}}
```

Store the original macros, and redefine accents.

```
\addto\extrasestonian{\babel@save\"\umlautlow\babel@save\~\tildelow}
```

Estonian does not use extra spaces after sentences.

```
\addto\extrasestonian{\bbl@frenchspacing}
\addto\noextrasestonian{\bbl@nonfrenchspacing}
```

\estonianhyphenmins

For Estonian, `\lefthyphenmin` and `\righthyphenmin` are both 2.

```
\providehyphenmins{\CurrentOption}{\tw@\tw@}
```

\tildelow
\gentilde
\newtilde
\newcheck

The standard TeX accents are too high for Estonian typography, we have to lower them (following the babel German style). For a detailed explanation see the file `glyphs.dtx`.

```
\def\tildelow{\def\~{\protect\gentilde}}
\def\gentilde#1{\if#1o\newtilde{#1}\else\if#1O\newtilde{#1}%
    \else\newcheck{#1}%
    \fi\fi}
\def\newtilde#1{\leavevmode\allowhyphens
  {\U@D 1ex%
  {\setbox\z@\hbox{\char126}\dimen@ -.45ex\advance\dimen@\ht\z@
  \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
  \accent126\fontdimen5\font\U@D #1}\allowhyphens}
\def\newcheck#1{\leavevmode\allowhyphens
  {\U@D 1ex%
  {\setbox\z@\hbox{\char20}\dimen@ -.45ex\advance\dimen@\ht\z@
  \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
  \accent20\fontdimen5\font\U@D #1}\allowhyphens}
```

We save the double quote character in `\dq`, and tilde in `\til`, and store the original definitions of `\"` and `\~` as `\dieresis` and `\texttilde`.

```
\begingroup \catcode`\"12
\edef\x{\endgroup
  \def\noexpand\dq{"}
  \def\noexpand\til{~}}
\x
\let\dieresis\"
\let\texttilde\~
```

This part follows closely `spanish.ldf`. We check the encoding and if it is T1, we have to tell TEX about our redefined accents.

```
\ifx\f@encoding\bbl@t@one
  \let\@umlaut\dieresis
  \let\@tilde\texttilde
  \DeclareTextComposite{\~}{T1}{s}{178}
  \DeclareTextComposite{\~}{T1}{S}{146}
  \DeclareTextComposite{\~}{T1}{z}{186}
  \DeclareTextComposite{\~}{T1}{Z}{154}
  \DeclareTextComposite{\"}{T1}{'}{17}
  \DeclareTextComposite{\"}{T1}{`}{18}
  \DeclareTextComposite{\"}{T1}{<}{19}
  \DeclareTextComposite{\"}{T1}{>}{20}
```

If the encoding differs from T1, we expand the accents, enabling hyphenation beyond the accent. In this case TEX will not find all possible breaks, and we have to warn people.

```
\else
  \wlog{Warning: Hyphenation would work better for the T1 encoding.}
  \let\@umlaut\newumlaut
  \let\@tilde\gentilde
\fi
```

Now we define the shorthands.

```
\declare@shorthand{estonian}{"a}{\textormath{\"{a}}{\ddot a}}
\declare@shorthand{estonian}{"A}{\textormath{\"{A}}{\ddot A}}
\declare@shorthand{estonian}{"o}{\textormath{\"{o}}{\ddot o}}
\declare@shorthand{estonian}{"O}{\textormath{\"{O}}{\ddot O}}
\declare@shorthand{estonian}{"u}{\textormath{\"{u}}{\ddot u}}
\declare@shorthand{estonian}{"U}{\textormath{\"{U}}{\ddot U}}
```

german and french quotes,

```
\declare@shorthand{estonian}{"`}{%
  \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
\declare@shorthand{estonian}{"'}{%
  \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
\declare@shorthand{estonian}{"<}{%
  \textormath{\guillemotleft}{\mbox{\guillemotleft}}}
\declare@shorthand{estonian}{">}{%
  \textormath{\guillemotright}{\mbox{\guillemotright}}}
```

```
\declare@shorthand{estonian}{~o}{\textormath{\@tilde o}{\tilde o}}
\declare@shorthand{estonian}{~O}{\textormath{\@tilde O}{\tilde O}}
\declare@shorthand{estonian}{~s}{\textormath{\@tilde s}{\check s}}
\declare@shorthand{estonian}{~S}{\textormath{\@tilde S}{\check S}}
\declare@shorthand{estonian}{~z}{\textormath{\@tilde z}{\check z}}
\declare@shorthand{estonian}{~Z}{\textormath{\@tilde Z}{\check Z}}
```

and some additional commands:

```
\declare@shorthand{estonian}{"-}{\nobreak\-\bbl@allowhyphens}
\declare@shorthand{estonian}{"|}{%
  \textormath{\nobreak\discretionary{-}{}{\kern.03em}%
              \allowhyphens}{}}
\declare@shorthand{estonian}{""}{\dq}
\declare@shorthand{estonian}{~~}{\til}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{estonian}
⟨/code⟩
```

## 48 The Croatian language

The file `croatian.dtx`[56] defines all the language definition macros for the Croatian language.

For this language currently no special definitions are needed or available.

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
⟨*code⟩
\LdfInit{croatian}\captionscroatian
```

When this file is read as an option, i.e. by the `\usepackage` command, `croatian` will be an 'unknown' language in which case we have to make it known. So we check for the existence of `\l@croatian` to see whether we have to do something here.

```
\ifx\l@croatian\@undefined
    \@nopatterns{Croatian}
    \adddialect\l@croatian0\fi
```

The next step consists of defining commands to switch to (and from) the Croatian language.

`\captionscroatian` The macro `\captionscroatian` defines all strings used in the four standard documentclasses provided with LaTeX.

```
\addto\captionscroatian{%
  \def\prefacename{Predgovor}%
  \def\refname{Literatura}%
  \def\abstractname{Sa\v{z}etak}%
  \def\bibname{Bibliografija}%
  \def\chaptername{Poglavlje}%
  \def\appendixname{Dodatak}%
  \def\contentsname{Sadr\v{z}aj}%
  \def\listfigurename{Popis slika}%
  \def\listtablename{Popis tablica}%
  \def\indexname{Indeks}%
  \def\figurename{Slika}%
  \def\tablename{Tablica}%
  \def\partname{Dio}%
  \def\enclname{Prilozi}%
  \def\ccname{Kopije}%
  \def\headtoname{Prima}%
  \def\pagename{Stranica}%
  \def\seename{Vidjeti}%
  \def\alsoname{Vidjeti i}%
  \def\proofname{Dokaz}%
  \def\glossaryname{Kazalo}%
  }%
```

[56]The file described in this section has version number v1.3l and was last revised on 2005/03/29. A contribution was made by Alan Paić (`paica@cernvm.cern.ch`).

`\datecroatian` The macro `\datecroatian` redefines the command `\today` to produce Croatian dates.

```
\def\datecroatian{%
  \def\today{\number\day.~\ifcase\month\or
    sije\v{c}nja\or velja\v{c}e\or o\v{z}ujka\or travnja\or svibnja\or
    lipnja\or srpnja\or kolovoza\or rujna\or listopada\or studenog\or
    prosinca\fi \space \number\year.}}
```

`\extrascroatian` `\noextrascroatian` The macro `\extrascroatian` will perform all the extra definitions needed for the Croatian language. The macro `\noextrascroatian` is used to cancel the actions of `\extrascroatian`. For the moment these macros are empty but they are defined for compatibility with the other language definition files.

```
\addto\extrascroatian{}
\addto\noextrascroatian{}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{croatian}
⟨/code⟩
```

## 49 The Czech language

The file `czech.dtx`[57] defines all the language definition macros for the Czech language.

For this language `\frenchspacing` is set and two macros `\q` and `\w` for easy access to two accents are defined.

The command `\q` is used with the letters (`t`, `d`, `l`, and `L`) and adds a `'` to them to simulate a 'hook' that should be there. The result looks like t'. The command `\w` is used to put the ring-accent which appears in ångstrøm over the letters `u` and `U`.

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
⟨*code⟩
\LdfInit{czech}\captionsczech
```

When this file is read as an option, i.e. by the `\usepackage` command, `czech` will be an 'unknown' language in which case we have to make it known. So we check for the existence of `\l@czech` to see whether we have to do something here.

```
\ifx\l@czech\@undefined
    \@nopatterns{Czech}
    \adddialect\l@czech0\fi
```

The next step consists of defining commands to switch to (and from) the Czech language.

`\captionsczech` The macro `\captionsczech` defines all strings used in the four standard documentclasses provided with LaTeX.

```
\addto\captionsczech{%
  \def\prefacename{P\v redmluva}%
  \def\refname{Reference}%
  \def\abstractname{Abstrakt}%
  \def\bibname{Literatura}%
  \def\chaptername{Kapitola}%
  \def\appendixname{Dodatek}%
  \def\contentsname{Obsah}%
  \def\listfigurename{Seznam obr\'azk\r{u}}%
  \def\listtablename{Seznam tabulek}%
  \def\indexname{Index}%
  \def\figurename{Obr\'azek}%
  \def\tablename{Tabulka}%
  \def\partname{\v{C}\'ast}%
  \def\enclname{P\v{r}\'{\i}loha}%
  \def\ccname{Na v\v{e}dom\'{\i}:}%
  \def\headtoname{Komu}%
  \def\pagename{Strana}%
  \def\seename{viz}%
```

[57]The file described in this section has version number v1.3k and was last revised on 2005/03/29. Contributions were made by Milos Lokajicek (LOKAJICK@CERNVM).

```
  \def\alsoname{viz tak\'e}%
  \def\proofname{D\r{u}kaz}%
  \def\glossaryname{Glos\'a\v r}%
  }%
```

`\dateczech` The macro `\dateczech` redefines the command `\today` to produce Czech dates.

```
\def\dateczech{%
  \def\today{\number\day.~\ifcase\month\or
    ledna\or \'unora\or b\v{r}ezna\or dubna\or kv\v{e}tna\or
    \v{c}ervna\or \v{c}ervence\or srpna\or z\'a\v{r}\'{\i}\or
    \v{r}\'{\i}jna\or listopadu\or prosince\fi
    \space \number\year}}
```

`\extrasczech` `\noextrasczech` The macro `\extrasczech` will perform all the extra definitions needed for the Czech language. The macro `\noextrasczech` is used to cancel the actions of `\extrasczech`. This means saving the meaning of two one-letter control sequences before defining them.

```
\addto\extrasczech{\babel@save\q\let\q\v}
\addto\extrasczech{\babel@save\w\let\w\r}
```

For Czech texts `\frenchspacing` should be in effect. We make sure this is the case and reset it if necessary.

```
\addto\extrasczech{\bbl@frenchspacing}
\addto\noextrasczech{\bbl@nonfrenchspacing}
```

`\v` LaTeX's normal `\v` accent places a caron over the letter that follows it (ǒ). This is not what we want for the letters d, t, l and L; for those the accent should change shape. This is acheived by the following.

```
\AtBeginDocument{%
  \DeclareTextCompositeCommand{\v}{OT1}{t}{%
    t\kern-.23em\raise.24ex\hbox{'}}
  \DeclareTextCompositeCommand{\v}{OT1}{d}{%
    d\kern-.13em\raise.24ex\hbox{'}}
  \DeclareTextCompositeCommand{\v}{OT1}{l}{\lcaron{}}
  \DeclareTextCompositeCommand{\v}{OT1}{L}{\Lcaron{}}}
```

`\lcaron` `\Lcaron` Fot the letters `l` and `L` we want to disinguish between normal fonts and monospaced fonts.

```
\def\lcaron{%
  \setbox0\hbox{M}\setbox\tw@\hbox{i}%
  \ifdim\wd0>\wd\tw@\relax
    l\kern-.13em\raise.24ex\hbox{'}\kern-.11em%
  \else
    l\raise.45ex\hbox to\z@{\kern-.35em '\hss}%
  \fi}
\def\Lcaron{%
  \setbox0\hbox{M}\setbox\tw@\hbox{i}%
  \ifdim\wd0>\wd\tw@\relax
    L\raise.24ex\hbox to\z@{\kern-.28em'\hss}%
  \else
    L\raise.45ex\hbox to\z@{\kern-.40em '\hss}%
  \fi}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{czech}
⟨/code⟩
```

## 50 The Polish language

The file `polish.dtx`[58] defines all the language-specific macros for the Polish language.

For this language the character `"` is made active. In table 22 an overview is given of its purpose.

| | |
|---|---|
| `"a` | or `\aob`, for tailed-a (like ą) |
| `"A` | or `\Aob`, for tailed-A (like Ą) |
| `"e` | or `\eob`, for tailed-e (like ę) |
| `"E` | or `\Eob`, for tailed-E (like Ę) |
| `"c` | or `\'c`, for accented c (like ć), same with uppercase letters and n,o,s |
| `"l` | or `\lpb{}`, for l with stroke (like ł) |
| `"L` | or `\Lpb{}`, for L with stroke (like Ł) |
| `"r` | or `\zkb{}`, for pointed z (like ż), cf. pronounciation |
| `"R` | or `\Zkb{}`, for pointed Z (like Ż) |
| `"z` | or `\'z`, for accented z |
| `"Z` | or `\'Z`, for accented Z |
| `"\|` | disable ligature at this position. |
| `"-` | an explicit hyphen sign, allowing hyphenation in the rest of the word. |
| `""` | like `"-`, but producing no hyphen sign (for compund words with hyphen, e.g. `x-""y`). |
| `"‘` | for German left double quotes (looks like „). |
| `"’` | for German right double quotes. |
| `"<` | for French left double quotes (similar to <<). |
| `">` | for French right double quotes (similar to >>). |

Table 22: The extra definitions made by `polish.sty`

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
50.1 ⟨*code⟩
50.2 \LdfInit{polish}\captionspolish
```

When this file is read as an option, i.e. by the `\usepackage` command, `polish` could be an ‘unknown’ language in which case we have to make it known. So we check for the existence of `\l@polish` to see whether we have to do something here.

```
50.3 \ifx\l@polish\@undefined
50.4   \@nopatterns{Polish}
50.5   \adddialect\l@polish0\fi
```

The next step consists of defining commands to switch to (and from) the Polish language.

[58]The file described in this section has version number v1.2l and was last revised on 2005/03/31.

\captionspolish The macro \captionspolish defines all strings used in the four standard documentclasses provided with LaTeX.

```
\addto\captionspolish{%
  \def\prefacename{Przedmowa}%
  \def\refname{Literatura}%
  \def\abstractname{Streszczenie}%
  \def\bibname{Bibliografia}%
  \def\chaptername{Rozdzia\l}%
  \def\appendixname{Dodatek}%
  \def\contentsname{Spis tre\'sci}%
  \def\listfigurename{Spis rysunk\'ow}%
  \def\listtablename{Spis tablic}%
  \def\indexname{Indeks}%
  \def\figurename{Rysunek}%
  \def\tablename{Tablica}%
  \def\partname{Cz\eob{}\'s\'c}%
  \def\enclname{Za\l\aob{}cznik}%
  \def\ccname{Kopie:}%
  \def\headtoname{Do}%
  \def\pagename{Strona}%
  \def\seename{Por\'ownaj}%
  \def\alsoname{Por\'ownaj tak\.ze}%
  \def\proofname{Dow\'od}%
  \def\glossaryname{Glossary}% <-- Needs translation
}
```

\datepolish The macro \datepolish redefines the command \today to produce Polish dates.

```
\def\datepolish{%
  \def\today{\number\day~\ifcase\month\or
  stycznia\or lutego\or marca\or kwietnia\or maja\or czerwca\or lipca\or
  sierpnia\or wrze\'snia\or pa\'zdziernika\or listopada\or grudnia\fi
  \space\number\year}%
}
```

\extraspolish \noextraspolish The macro \extraspolish will perform all the extra definitions needed for the Polish language. The macro \noextraspolish is used to cancel the actions of \extraspolish.

For Polish the " character is made active. This is done once, later on its definition may vary. Other languages in the same document may also use the " character for shorthands; we specify that the polish group of shorthands should be used.

```
\initiate@active@char{"}
\addto\extraspolish{\languageshorthands{polish}}
\addto\extraspolish{\bbl@activate{"}}
```

Don't forget to turn the shorthands off again.

```
\addto\noextraspolish{\bbl@deactivate{"}}
```

The code above is necessary because we need an extra active character. This character is then used as indicated in table 22.

If you have problems at the end of a word with a linebreak, use the other version without hyphenation tricks. Some TeX wizard may produce a better solution with forcasting another token to decide whether the character after the double quote is the last in a word. Do it and let us know.

In Polish texts some letters get special diacritical marks. Leszek Holenderski designed the following code to position the diacritics correctly for every font in every size. These macros need a few extra dimension variables.

```
\newdimen\pl@left
\newdimen\pl@down
\newdimen\pl@right
\newdimen\pl@temp
```

`\sob` The macro `\sob` is used to put the 'ogonek' in the right place.

```
\def\sob#1#2#3#4#5{%parameters: letter and fractions hl,ho,vl,vo
  \setbox0\hbox{#1}\setbox1\hbox{$_\mathchar'454$}\setbox2\hbox{p}%
  \pl@right=#2\wd0 \advance\pl@right by-#3\wd1
  \pl@down=#5\ht1 \advance\pl@down by-#4\ht0
  \pl@left=\pl@right \advance\pl@left by\wd1
  \pl@temp=-\pl@down \advance\pl@temp by\dp2 \dp1=\pl@temp
  \leavevmode
  \kern\pl@right\lower\pl@down\box1\kern-\pl@left #1}
```

`\aob` `\Aob` `\eob` `\Eob` The ogonek is placed with the letters 'a', 'A', 'e', and 'E'.

```
\DeclareTextCommand{\aob}{OT1}{\sob a{.66}{.20}{0}{.90}}
\DeclareTextCommand{\Aob}{OT1}{\sob A{.80}{.50}{0}{.90}}
\DeclareTextCommand{\eob}{OT1}{\sob e{.50}{.35}{0}{.93}}
\DeclareTextCommand{\Eob}{OT1}{\sob E{.60}{.35}{0}{.90}}
```

For the 'new' `T1` encoding we can provide simpler definitions.

```
\DeclareTextCommand{\aob}{T1}{\k a}
\DeclareTextCommand{\Aob}{T1}{\k A}
\DeclareTextCommand{\eob}{T1}{\k e}
\DeclareTextCommand{\Eob}{T1}{\k E}
```

Construct the characters by default from the OT1 encoding.

```
\ProvideTextCommandDefault{\aob}{\UseTextSymbol{OT1}{\aob}}
\ProvideTextCommandDefault{\Aob}{\UseTextSymbol{OT1}{\Aob}}
\ProvideTextCommandDefault{\eob}{\UseTextSymbol{OT1}{\eob}}
\ProvideTextCommandDefault{\Eob}{\UseTextSymbol{OT1}{\Eob}}
```

`\spb` The macro `\spb` is used to put the 'poprzeczka' in the right place.

```
\def\spb#1#2#3#4#5{%
  \setbox0\hbox{#1}\setbox1\hbox{\char'023}%
  \pl@right=#2\wd0 \advance\pl@right by-#3\wd1
  \pl@down=#5\ht1 \advance\pl@down by-#4\ht0
  \pl@left=\pl@right \advance\pl@left by\wd1
  \ht1=\pl@down \dp1=-\pl@down
  \leavevmode
  \kern\pl@right\lower\pl@down\box1\kern-\pl@left #1}
```

\skb The macro `\skb` is used to put the 'kropka' in the right place.

```
\def\skb#1#2#3#4#5{%
  \setbox0\hbox{#1}\setbox1\hbox{\char'056}%
  \pl@right=#2\wd0 \advance\pl@right by-#3\wd1
  \pl@down=#5\ht1 \advance\pl@down by-#4\ht0
  \pl@left=\pl@right \advance\pl@left by\wd1
  \leavevmode
  \kern\pl@right\lower\pl@down\box1\kern-\pl@left #1}
```

\textpl For the 'poprzeczka' and the 'kropka' in text fonts we don't need any special coding, but we can (almost) use what is already available.

```
\def\textpl{%
  \def\lpb{\plll}%
  \def\Lpb{\pLLL}%
  \def\zkb{\.z}%
  \def\Zkb{\.Z}}
```

Initially we assume that typesetting is done with text fonts.

```
\textpl
```

```
\let\lll=\l \let\LLL=\L
\def\plll{\lll}
\def\pLLL{\LLL}
```

\telepl But for the 'teletype' font in 'OT1' encoding we have to take some special actions, involving the macros defined above.

```
\def\telepl{%
  \def\lpb{\spb l{.45}{.5}{.4}{.8}}%
  \def\Lpb{\spb L{.23}{.5}{.4}{.8}}%
  \def\zkb{\skb z{.5}{.5}{1.2}{0}}%
  \def\Zkb{\skb Z{.5}{.5}{1.1}{0}}}
```

To activate these codes the font changing commands as they are defined in LaTeX are modified. The same is done for plain TeX's font changing commands.

When `\selectfont` is undefined the current format is spposed to be either plain (based) or LaTeX 2.09.

```
\ifx\selectfont\@undefined
  \ifx\prm\@undefined \addto\rm{\textpl}\else \addto\prm{\textpl}\fi
  \ifx\pit\@undefined \addto\it{\textpl}\else \addto\pit{\textpl}\fi
  \ifx\pbf\@undefined \addto\bf{\textpl}\else \addto\pbf{\textpl}\fi
  \ifx\psl\@undefined \addto\sl{\textpl}\else \addto\psl{\textpl}\fi
  \ifx\psf\@undefined                   \else \addto\psf{\textpl}\fi
  \ifx\psc\@undefined                   \else \addto\psc{\textpl}\fi
  \ifx\ptt\@undefined \addto\tt{\telepl}\else \addto\ptt{\telepl}\fi
\else
```

When `\selectfont` exists we assume $\LaTeX\,2_\varepsilon$.

```
  \expandafter\addto\csname selectfont \endcsname{%
    \csname\f@encoding @pl\endcsname}
\fi
```

Currently we support the OT1 and T1 encodings. For T1 we don't have to make a difference between typewriter fonts and other fonts, they all have the same glyphs.

```
\expandafter\let\csname T1@pl\endcsname\textpl
```

For OT1 we need to check the current font family, stored in `\f@family`. Unfortunately we need a hack as `\ttdefault` is defined as a `\long` macro, while `\f@family` is not.

```
\expandafter\def\csname OT1@pl\endcsname{%
  \long\edef\curr@family{\f@family}%
  \ifx\curr@family\ttdefault
    \telepl
  \else
    \textpl
  \fi}
```

`\dq` We save the original double quote character in `\dq` to keep it available, the math accent `\"` can now be typed as `"`.

```
\begingroup \catcode`\"12
\def\x{\endgroup
  \def\dq{"}}
\x
```

Now we can define the doublequote macros for diacritics,

```
\declare@shorthand{polish}{"a}{\textormath{\aob}{\ddot a}}
\declare@shorthand{polish}{"A}{\textormath{\Aob}{\ddot A}}
\declare@shorthand{polish}{"c}{\textormath{\'c}{\acute c}}
\declare@shorthand{polish}{"C}{\textormath{\'C}{\acute C}}
\declare@shorthand{polish}{"e}{\textormath{\eob}{\ddot e}}
\declare@shorthand{polish}{"E}{\textormath{\Eob}{\ddot E}}
\declare@shorthand{polish}{"l}{\textormath{\lpb}{\ddot l}}
\declare@shorthand{polish}{"L}{\textormath{\Lpb}{\ddot L}}
\declare@shorthand{polish}{"n}{\textormath{\'n}{\acute n}}
\declare@shorthand{polish}{"N}{\textormath{\'N}{\acute N}}
\declare@shorthand{polish}{"o}{\textormath{\'o}{\acute o}}
\declare@shorthand{polish}{"O}{\textormath{\'O}{\acute O}}
\declare@shorthand{polish}{"s}{\textormath{\'s}{\acute s}}
\declare@shorthand{polish}{"S}{\textormath{\'S}{\acute S}}
```

`\polishrz`
`\polishzx` The command `\polishrz` defines the shorthands `"r`, `"z` and `"x` to produce pointed z, accented z and `"x`. This is the default as these shorthands were defined by this language definition file for quite some time.

```
\newcommand*{\polishrz}{%
  \declare@shorthand{polish}{"r}{\textormath{\zkb}{\ddot r}}%
  \declare@shorthand{polish}{"R}{\textormath{\Zkb}{\ddot R}}%
  \declare@shorthand{polish}{"z}{\textormath{\'z}{\acute z}}%
  \declare@shorthand{polish}{"Z}{\textormath{\'Z}{\acute Z}}%
  \declare@shorthand{polish}{"x}{\dq x}%
  \declare@shorthand{polish}{"X}{\dq X}%
  }
\polishrz
```

The command `\polishzx` switches to a different set of shorthands, `"z`, `"x` and `"r` to produce pointed z, accented z and `"r`; a different shorthand notation also in use.

```
\newcommand*{\polishzx}{%
  \declare@shorthand{polish}{"z}{\textormath{\zkb}{\ddot z}}%
  \declare@shorthand{polish}{"Z}{\textormath{\Zkb}{\ddot Z}}%
  \declare@shorthand{polish}{"x}{\textormath{\'z}{\acute x}}%
  \declare@shorthand{polish}{"X}{\textormath{\'Z}{\acute X}}%
  \declare@shorthand{polish}{"r}{\dq r}%
  \declare@shorthand{polish}{"R}{\dq R}%
  }
```

Then we define access to two forms of quotation marks, similar to the german and french quotation marks.

```
\declare@shorthand{polish}{"`}{%
  \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
\declare@shorthand{polish}{"'}{%
  \textormath{\textquotedblright}{\mbox{\textquotedblright}}}
\declare@shorthand{polish}{"<}{%
  \textormath{\guillemotleft}{\mbox{\guillemotleft}}}
\declare@shorthand{polish}{">}{%
  \textormath{\guillemotright}{\mbox{\guillemotright}}}
```

then we define two shorthands to be able to specify hyphenation breakpoints that behavew a little different from `\-`.

```
\declare@shorthand{polish}{"-}{\nobreak-\bbl@allowhyphens}
\declare@shorthand{polish}{""}{\hskip\z@skip}
```

And we want to have a shorthand for disabling a ligature.

```
\declare@shorthand{polish}{"|}{%
  \textormath{\discretionary{-}{}{\kern.03em}}{}}
```

`\mdqon` `\mdqoff` All that's left to do now is to define a couple of commands for reasons of compatibility with `polish.tex`.

```
\def\mdqon{\shorthandon{"}}
\def\mdqoff{\shorthandoff{"}}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{polish}
⟨/code⟩
```

## 51 The Serbocroatian language

The file `serbian.dtx`[59] defines all the language definition macros for the Serbian language, typeset in a latin script. In a future version support for typesetting in a cyrillic script may be added.

For this language the character `"` is made active. In table 23 an overview is given of its purpose. One of the reasons for this is that in the Serbian language some special characters are used.

| | |
|---|---|
| `"c` | `\"c`, also implemented for the lowercase and uppercase s and z. |
| `"d` | `\dj`, also implemented for `"D` |
| `"-` | an explicit hyphen sign, allowing hyphenation in the rest of the word. |
| `"\|` | disable ligature at this position |
| `""` | like `"-`, but producing no hyphen sign (for compund words with hyphen, e.g. `x-""y`). |
| `"‘` | for Serbian left double quotes (looks like „). |
| `"’` | for Serbian right double quotes. |
| `"<` | for French left double quotes (similar to <<). |
| `">` | for French right double quotes (similar to >>). |

Table 23: The extra definitions made by `serbian.ldf`

Apart from defining shorthands we need to make sure taht the first paragraph of each section is intended. Furthermore the following new math operators are defined (`\tg`, `\ctg`, `\arctg`, `\arcctg`, `\sh`, `\ch`, `\th`, `\cth`, `\arsh`, `\arch`, `\arth`, `\arcth`, `\Prob`, `\Expect`, `\Variance`).

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
51.1 ⟨*code⟩
51.2 \LdfInit{serbian}\captionsserbian
```

When this file is read as an option, i.e. by the `\usepackage` command, `serbian` will be an ‘unknown’ language in which case we have to make it known. So we check for the existence of `\l@serbian` to see whether we have to do something here.

```
51.3 \ifx\l@serbian\@undefined
51.4     \@nopatterns{Serbian}
51.5     \adddialect\l@serbian0\fi
```

The next step consists of defining commands to switch to (and from) the Serbocroatian language.

[59]The file described in this section has version number v1.0d and was last revised on 2005/03/31. A contribution was made by Dejan Muhamedagić (`dejan@yunix.com`).

\captionsserbian The macro `\captionsserbian` defines all strings used in the four standard documentclasses provided with LaTeX.

```
\addto\captionsserbian{%
  \def\prefacename{Predgovor}%
  \def\refname{Literatura}%
  \def\abstractname{Sa\v{z}etak}%
  \def\bibname{Bibliografija}%
  \def\chaptername{Glava}%
  \def\appendixname{Dodatak}%
  \def\contentsname{Sadr\v{z}aj}%
  \def\listfigurename{Slike}%
  \def\listtablename{Tabele}%
  \def\indexname{Indeks}%
  \def\figurename{Slika}%
  \def\tablename{Tabela}%
  \def\partname{Deo}%
  \def\enclname{Prilozi}%
  \def\ccname{Kopije}%
  \def\headtoname{Prima}%
  \def\pagename{Strana}%
  \def\seename{Vidi}%
  \def\alsoname{Vidi tako\dj e}%
  \def\proofname{Dokaz}%
  \def\glossaryname{Glossary}% <-- Needs translation
  }%
```

\dateserbian The macro `\dateserbian` redefines the command `\today` to produce Serbocroatian dates.

```
\def\dateserbian{%
  \def\today{\number\day .~\ifcase\month\or
    januar\or februar\or mart\or april\or maj\or
    juni\or juli\or avgust\or septembar\or oktobar\or novembar\or
    decembar\fi \space \number\year}}
```

\extrasserbian \noextrasserbian The macro `\extrasserbian` will perform all the extra definitions needed for the Serbocroatian language. The macro `\noextrasserbian` is used to cancel the actions of `\extrasserbian`.

For Serbian the `"` character is made active. This is done once, later on its definition may vary. Other languages in the same document may also use the `"` character for shorthands; we specify that the serbian group of shorthands should be used.

```
\initiate@active@char{"}
\addto\extrasserbian{\languageshorthands{serbian}}
\addto\extrasserbian{\bbl@activate{"}}
```

Don't forget to turn the shorthands off again.

```
\addto\noextrasserbian{\bbl@deactivate{"}}
```

First we define shorthands to facilitate the occurence of letters such as č.

```
\declare@shorthand{serbian}{"c}{\textormath{\v c}{\check c}}
\declare@shorthand{serbian}{"d}{\textormath{\dj}{\dj}}%%
\declare@shorthand{serbian}{"s}{\textormath{\v s}{\check s}}
\declare@shorthand{serbian}{"z}{\textormath{\v z}{\check z}}
\declare@shorthand{serbian}{"C}{\textormath{\v C}{\check C}}
\declare@shorthand{serbian}{"D}{\textormath{\DJ}{\DJ}}%%
\declare@shorthand{serbian}{"S}{\textormath{\v S}{\check S}}
\declare@shorthand{serbian}{"Z}{\textormath{\v Z}{\check Z}}
```

Then we define access to two forms of quotation marks, similar to the german and french quotation marks.

```
\declare@shorthand{serbian}{"`}{%
  \textormath{\quotedblbase{}}{\mbox{\quotedblbase}}}
\declare@shorthand{serbian}{"'}{%
  \textormath{\textquotedblleft{}}{\mbox{\textquotedblleft}}}
\declare@shorthand{serbian}{"<}{%
  \textormath{\guillemotleft{}}{\mbox{\guillemotleft}}}
\declare@shorthand{serbian}{">}{%
  \textormath{\guillemotright{}}{\mbox{\guillemotright}}}
```

then we define two shorthands to be able to specify hyphenation breakpoints that behave a little different from `\-`.

```
\declare@shorthand{serbian}{"-}{\nobreak-\bbl@allowhyphens}
\declare@shorthand{serbian}{""}{\hskip\z@skip}
```

And we want to have a shorthand for disabling a ligature.

```
\declare@shorthand{serbian}{"|}{%
  \textormath{\discretionary{-}{}{\kern.03em}}{}}
```

`\bbl@frenchindent`
`\bbl@nonfrenchindent`

In Serbian the first paragraph of each section should be indented. Add this code only in LaTeX.

```
\ifx\fmtname plain \else
  \let\@aifORI\@afterindentfalse
  \def\bbl@frenchindent{\let\@afterindentfalse\@afterindenttrue
                         \@afterindenttrue}
  \def\bbl@nonfrenchindent{\let\@afterindentfalse\@aifORI
                            \@afterindentfalse}
  \addto\extrasserbian{\bbl@frenchindent}
  \addto\noextrasserbian{\bbl@nonfrenchindent}
\fi
```

`\mathserbian`

Some math functions in Serbian math books have other names: e.g. `sinh` in Serbian is written as `sh` etc. So we define a number of new math operators.

```
\def\sh{\mathop{\operator@font sh}\nolimits} % same as \sinh
\def\ch{\mathop{\operator@font ch}\nolimits} % same as \cosh
\def\th{\mathop{\operator@font th}\nolimits} % same as \tanh
\def\cth{\mathop{\operator@font cth}\nolimits} % same as \coth
\def\arsh{\mathop{\operator@font arsh}\nolimits}
\def\arch{\mathop{\operator@font arch}\nolimits}
\def\arth{\mathop{\operator@font arth}\nolimits}
```

```
\def\arcth{\mathop{\operator@font arcth}\nolimits}
\def\tg{\mathop{\operator@font tg}\nolimits} % same as \tan
\def\ctg{\mathop{\operator@font ctg}\nolimits} % same as \cot
\def\arctg{\mathop{\operator@font arctg}\nolimits} % same as \arctan
\def\arcctg{\mathop{\operator@font arcctg}\nolimits}
\def\Prob{\mathop{\mathsf P\hskip0pt}\nolimits}
\def\Expect{\mathop{\mathsf E\hskip0pt}\nolimits}
\def\Variance{\mathop{\mathsf D\hskip0pt}\nolimits}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{serbian}
⟨/code⟩
```

## 52 The Slovak language

The file `slovak.dtx`[60] defines all the language-specific macros for the Slovak language.

For this language the macro `\q` is defined. It was used with the letters (`t`, `d`, `l`, and `L`) and adds a `'` to them to simulate a 'hook' that should be there. The result looks like t'. Since the the T1 font encoding has the corresponding characters it is mapped to `\v`. Therefore we recommend using T1 font encoding. If you don't want to use this encoding, please, feel free to redefine `\q` in your file. I think babel will honour this `;-)`.

For this language the characters `"`, `'` and `^` are ade active. In table 24 an overview is given of its purpose. Also the vertical placement of the umlaut can be controlled this way.

| | |
|---|---|
| `"a` | `\"a`, also implemented for the other lowercase and uppercase vowels. |
| `^d` | `\q d`, also implemented for l, t and L. |
| `^c` | `\v c`, also implemented for C, D, N, n, T, Z and z. |
| `^o` | `\^o`, also implemented for O. |
| `'a` | `\'a`, also implemented for the other lowercase and uppercase l, r, y and vowels. |
| `"\|` | disable ligature at this position. |
| `"-` | an explicit hyphen sign, allowing hyphenation in the rest of the word. |
| `""` | like `"-`, but producing no hyphen sign (for compund words with hyphen, e.g. `x-""y`). |
| `"~` | for a compound word mark without a breakpoint. |
| `"=` | for a compound word mark with a breakpoint, allowing hyphenation in the composing words. |
| `"‘` | for German left double quotes (looks like „). |
| `"’` | for German right double quotes. |
| `"<` | for French left double quotes (similar to <<). |
| `">` | for French right double quotes (similar to >>). |

Table 24: The extra definitions made by `slovak.ldf`

The quotes in table 24 can also be typeset by using the commands in table 25.

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
52.1 ⟨*code⟩
52.2 \LdfInit{slovak}\captionsslovak
```

[60]The file described in this section has version number v1.3a and was last revised on 2005/03/31. It was written by Jana Chlebikova (`chlebik@euromath.dk`) and modified by Tobias Schlemmer (`Tobias.Schlemmer@web.de`).

| | |
|---|---|
| `\glqq` | for German left double quotes (looks like „). |
| `\grqq` | for German right double quotes (looks like “). |
| `\glq` | for German left single quotes (looks like ‚). |
| `\grq` | for German right single quotes (looks like ‘). |
| `\flqq` | for French left double quotes (similar to <<). |
| `\frqq` | for French right double quotes (similar to >>). |
| `\flq` | for (French) left single quotes (similar to <). |
| `\frq` | for (French) right single quotes (similar to >). |
| `\dq` | the original quotes character ("). |
| `\sq` | the original single quote ('). |

Table 25: More commands which produce quotes, defined by `slovak.ldf`

When this file is read as an option, i.e. by the `\usepackage` command, `slovak` will be an 'unknown' language in which case we have to make it known. So we check for the existence of `\l@slovak` to see whether we have to do something here.

```
52.3 \ifx\l@slovak\@undefined
52.4     \@nopatterns{Slovak}
52.5     \adddialect\l@slovak0\fi
```

The next step consists of defining commands to switch to (and from) the Slovak language.

`\captionsslovak` The macro `\captionsslovak` defines all strings used in the four standard documentclasses provided with LaTeX.

```
52.6 \addto\captionsslovak{%
52.7   \def\prefacename{\'Uvod}%
52.8   \def\refname{Referencie}%
52.9   \def\abstractname{Abstrakt}%
52.10  \def\bibname{Literat\'ura}%
52.11  \def\chaptername{Kapitola}%
52.12  \def\appendixname{Dodatok}%
52.13  \def\contentsname{Obsah}%
52.14  \def\listfigurename{Zoznam obr\'azkov}%
52.15  \def\listtablename{Zoznam tabuliek}%
52.16  \def\indexname{Index}%
52.17  \def\figurename{Obr\'azok}%
52.18  \def\tablename{Tabu\q lka}%%% special letter l with hook
52.19  \def\partname{\v{C}as\q t}%%% special letter t with hook
52.20  \def\enclname{Pr\'{\i}lohy}%
52.21  \def\ccname{CC}%
52.22  \def\headtoname{Komu}%
52.23  \def\pagename{Strana}%
52.24  \def\seename{vi\q d}%%%  Special letter d with hook
52.25  \def\alsoname{vi\q d tie\v z}%%%  Special letter d with hook
52.26  \def\proofname{D\^okaz}%
52.27  \def\glossaryname{Glossary}% <-- Needs translation
```

```
  }
```

\dateslovak The macro \dateslovak redefines the command \today to produce Slovak dates.

```
\def\dateslovak{%
  \def\today{\number\day.~\ifcase\month\or
    janu\'ara\or febru\'ara\or marca\or apr\'{\i}la\or m\'aja\or
    j\'una\or j\'ula\or augusta\or septembra\or okt\'obra\or
    novembra\or decembra\fi
    \space \number\year}}
```

\extrasslovak \noextrasslovak The macro \extrasslovak will perform all the extra definitions needed for the Slovak language. The macro \noextrasslovak is used to cancel the actions of \extrasslovak. For Slovak three characters are used to define shorthands, they need to be made active.

```
\addto\extrasslovak{\languageshorthands{slovak}}
\initiate@active@char{^}
\addto\extrasslovak{\bbl@activate{^}}
\addto\noextrasslovak{\bbl@deactivate{^}}
\initiate@active@char{"}
\addto\extrasslovak{\bbl@activate{"}\umlautlow}
\addto\noextrasslovak{\bbl@deactivate{"}\umlauthigh}
\initiate@active@char{'}
\@ifpackagewith{babel}{activeacute}{%
  \addto\extrasslovak{\bbl@activate{'}}
  \addto\noextrasslovak{\bbl@deactivate{'}}%
  }{}
```

```
\addto\extrasslovak{\babel@save\q\let\q\v}
```

The slovak hyphenation patterns should be used with \lefthyphenmin set to 2 and \righthyphenmin set to 2.

```
\providehyphenmins{\CurrentOption}{\tw@\tw@}
```

\dq We save the original double quote character in \dq to keep it available, the math accent \" can now be typed as ".

```
\begingroup \catcode`\"12
\def\x{\endgroup
  \def\sq{'}
  \def\dq{"}}
\x
```

In order to prevent problems with the active ^ we add a shorthand on system level which expands to a 'normal ^.

```
\declare@shorthand{system}{^}{\csname normal@char\string^\endcsname}
```

Now we can define the doublequote macros: the umlauts,

```
\declare@shorthand{slovak}{"a}{\textormath{\"{a}\allowhyphens}{\ddot a}}
\declare@shorthand{slovak}{"o}{\textormath{\"{o}\allowhyphens}{\ddot o}}
```

```
\declare@shorthand{slovak}{"u}{\textormath{\"{u}\allowhyphens}{\ddot u}}
\declare@shorthand{slovak}{"A}{\textormath{\"{A}\allowhyphens}{\ddot A}}
\declare@shorthand{slovak}{"O}{\textormath{\"{O}\allowhyphens}{\ddot O}}
\declare@shorthand{slovak}{"U}{\textormath{\"{U}\allowhyphens}{\ddot U}}
```

tremas,

```
\declare@shorthand{slovak}{"e}{\textormath{\"{e}\allowhyphens}{\ddot e}}
\declare@shorthand{slovak}{"E}{\textormath{\"{E}\allowhyphens}{\ddot E}}
\declare@shorthand{slovak}{"i}{\textormath{\"{\i}\allowhyphens}%
                                 {\ddot\imath}}
\declare@shorthand{slovak}{"I}{\textormath{\"{I}\allowhyphens}{\ddot I}}
```

other slovak characters

```
\declare@shorthand{slovak}{^c}{\textormath{\v{c}\allowhyphens}{\check{c}}}
\declare@shorthand{slovak}{^d}{\textormath{\q{d}\allowhyphens}{\check{d}}}
\declare@shorthand{slovak}{^l}{\textormath{\q{l}\allowhyphens}{\check{l}}}
\declare@shorthand{slovak}{^n}{\textormath{\v{n}\allowhyphens}{\check{n}}}
\declare@shorthand{slovak}{^o}{\textormath{\^{o}\allowhyphens}{\hat{o}}}
\declare@shorthand{slovak}{^s}{\textormath{\v{s}\allowhyphens}{\check{s}}}
\declare@shorthand{slovak}{^t}{\textormath{\q{t}\allowhyphens}{\check{t}}}
\declare@shorthand{slovak}{^z}{\textormath{\v{z}\allowhyphens}{\check{z}}}
\declare@shorthand{slovak}{^C}{\textormath{\v{C}\allowhyphens}{\check{C}}}
\declare@shorthand{slovak}{^D}{\textormath{\v{D}\allowhyphens}{\check{D}}}
\declare@shorthand{slovak}{^L}{\textormath{\q{L}\allowhyphens}{\check{L}}}
\declare@shorthand{slovak}{^N}{\textormath{\v{N}\allowhyphens}{\check{N}}}
\declare@shorthand{slovak}{^O}{\textormath{\^{O}\allowhyphens}{\hat{O}}}
\declare@shorthand{slovak}{^S}{\textormath{\v{S}\allowhyphens}{\check{S}}}
\declare@shorthand{slovak}{^T}{\textormath{\v{T}\allowhyphens}{\check{T}}}
\declare@shorthand{slovak}{^Z}{\textormath{\v{Z}\allowhyphens}{\check{Z}}}
```

acute accents,

```
\@ifpackagewith{babel}{activeacute}{%
  \declare@shorthand{slovak}{'a}{\textormath{\'a\allowhyphens}{^{\prime}a}}
  \declare@shorthand{slovak}{'e}{\textormath{\'e\allowhyphens}{^{\prime}e}}
  \declare@shorthand{slovak}{'i}{\textormath{\'\i{}\allowhyphens}{^{\prime}i}}
  \declare@shorthand{slovak}{'l}{\textormath{\'l\allowhyphens}{^{\prime}l}}
  \declare@shorthand{slovak}{'o}{\textormath{\'o\allowhyphens}{^{\prime}o}}
  \declare@shorthand{slovak}{'r}{\textormath{\'r\allowhyphens}{^{\prime}r}}
  \declare@shorthand{slovak}{'u}{\textormath{\'u\allowhyphens}{^{\prime}u}}
  \declare@shorthand{slovak}{'y}{\textormath{\'y\allowhyphens}{^{\prime}y}}
  \declare@shorthand{slovak}{'A}{\textormath{\'A\allowhyphens}{^{\prime}A}}
  \declare@shorthand{slovak}{'E}{\textormath{\'E\allowhyphens}{^{\prime}E}}
  \declare@shorthand{slovak}{'I}{\textormath{\'I\allowhyphens}{^{\prime}I}}
  \declare@shorthand{slovak}{'L}{\textormath{\'L\allowhyphens}{^{\prime}l}}
  \declare@shorthand{slovak}{'O}{\textormath{\'O\allowhyphens}{^{\prime}O}}
  \declare@shorthand{slovak}{'R}{\textormath{\'R\allowhyphens}{^{\prime}R}}
  \declare@shorthand{slovak}{'U}{\textormath{\'U\allowhyphens}{^{\prime}U}}
  \declare@shorthand{slovak}{'Y}{\textormath{\'Y\allowhyphens}{^{\prime}Y}}
  \declare@shorthand{slovak}{''}{%
    \textormath{\textquotedblright}{\sp\bgroup\prim@s'}}
  }{}
```

german and french quotes,

```
\declare@shorthand{slovak}{"`}{\glqq}
\declare@shorthand{slovak}{"'}{\grqq}
\declare@shorthand{slovak}{"<}{\flqq}
\declare@shorthand{slovak}{">}{\frqq}
```

and some additional commands:

```
\declare@shorthand{slovak}{"-}{\nobreak\-\bbl@allowhyphens}
\declare@shorthand{slovak}{"|}{%
  \textormath{\penalty\@M\discretionary{-}{}{\kern.03em}%
              \bbl@allowhyphens}{}}
\declare@shorthand{slovak}{""}{\hskip\z@skip}
\declare@shorthand{slovak}{"~}{\textormath{\leavevmode\hbox{-}}{-}}
\declare@shorthand{slovak}{"=}{\nobreak-\hskip\z@skip}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{slovak}
⟨/code⟩
```

## 53 The Slovenian language

The file `slovene.dtx`[61] defines all the language-specific macros for the Slovenian language.

For this language the character `"` is made active. In table 26 an overview is given of its purpose. One of the reasons for this is that in the Slovene language some special characters are used.

| | |
|---|---|
| `"c` | `\"c`, also implemented for the lowercase and uppercase s and z. |
| `"-` | an explicit hyphen sign, allowing hyphenation in the rest of the word. |
| `""` | like `"-`, but producing no hyphen sign (for compund words with hyphen, e.g. `x-""y`). |
| `"‘` | for Slovene left double quotes (looks like „). |
| `"’` | for Slovene right double quotes. |
| `"<` | for French left double quotes (similar to <<). |
| `">` | for French right double quotes (similar to >>). |

Table 26: The extra definitions made by `slovene.ldf`

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
⟨*code⟩
\LdfInit{slovene}\captionsslovene
```

When this file is read as an option, i.e. by the `\usepackage` command, `slovene` will be an ‘unknown’ language in which case we have to make it known. So we check for the existence of `\l@slovene` to see whether we have to do something here.

```
\ifx\l@slovene\@undefined
    \@nopatterns{Slovene}
    \adddialect\l@slovene0\fi
```

The next step consists of defining commands to switch to the Slovenian language. The reason for this is that a user might want to switch back and forth between languages.

`\captionsslovene` The macro `\captionsslovene` defines all strings used in the four standard documentclasses provided with LaTeX.

```
\addto\captionsslovene{%
  \def\prefacename{Predgovor}%
  \def\refname{Literatura}%
  \def\abstractname{Povzetek}%
```

[61]The file described in this section has version number v1.2m and was last revised on 2005/03/31. Contributions were made by Danilo Zavrtanik, University of Ljubljana (YU) and Leon Žlajpah (`leon.zlajpah@ijs.si`).

```
  \def\bibname{Literatura}%
  \def\chaptername{Poglavje}%
  \def\appendixname{Dodatek}%
  \def\contentsname{Kazalo}%
  \def\listfigurename{Slike}%
  \def\listtablename{Tabele}%
  \def\indexname{Stvarno kazalo}% used to be Indeks
  \def\figurename{Slika}%
  \def\tablename{Tabela}%
  \def\partname{Del}%
  \def\enclname{Priloge}%
  \def\ccname{Kopije}%
  \def\headtoname{Prejme}%
  \def\pagename{Stran}%
  \def\seename{glej}%
  \def\alsoname{glej tudi}%
  \def\proofname{Dokaz}%
  \def\glossaryname{Glossary}% <-- Needs translation
  }%
```

`\dateslovene` The macro `\dateslovene` redefines the command `\today` to produce Slovenian dates.

```
\def\dateslovene{%
  \def\today{\number\day.~\ifcase\month\or
    januar\or februar\or marec\or april\or maj\or junij\or
    julij\or avgust\or september\or oktober\or november\or december\fi
    \space \number\year}}
```

`\extrasslovene` `\noextrasslovene` The macro `\extrasslovene` performs all the extra definitions needed for the Slovenian language. The macro `\noextrasslovene` is used to cancel the actions of `\extrasslovene`.

For Slovene the " character is made active. This is done once, later on its definition may vary. Other languages in the same document may also use the " character for shorthands; we specify that the slovenian group of shorthands should be used.

```
\initiate@active@char{"}
\addto\extrasslovene{\languageshorthands{slovene}}
\addto\extrasslovene{\bbl@activate{"}}
```

Don't forget to turn the shorthands off again.

```
\addto\noextrasslovene{\bbl@deactivate{"}}
```

First we define shorthands to facilitate the occurence of letters such as č.

```
\declare@shorthand{slovene}{"c}{\textormath{\v c}{\check c}}
\declare@shorthand{slovene}{"s}{\textormath{\v s}{\check s}}
\declare@shorthand{slovene}{"z}{\textormath{\v z}{\check z}}
\declare@shorthand{slovene}{"C}{\textormath{\v C}{\check C}}
\declare@shorthand{slovene}{"S}{\textormath{\v S}{\check S}}
\declare@shorthand{slovene}{"Z}{\textormath{\v Z}{\check Z}}
```

Then we define access to two forms of quotation marks, similar to the german and french quotation marks.

```
\declare@shorthand{slovene}{"`}{%
  \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
\declare@shorthand{slovene}{"'}{%
  \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
\declare@shorthand{slovene}{"<}{%
  \textormath{\guillemotleft}{\mbox{\guillemotleft}}}
\declare@shorthand{slovene}{">}{%
  \textormath{\guillemotright}{\mbox{\guillemotright}}}
```

then we define two shorthands to be able to specify hyphenation breakpoints that behavew a little different from `\-`.

```
\declare@shorthand{slovene}{"-}{\nobreak-\bbl@allowhyphens}
\declare@shorthand{slovene}{""}{\hskip\z@skip}
```

And we want to have a shorthand for disabling a ligature.

```
\declare@shorthand{slovene}{"|}{%
  \textormath{\discretionary{-}{}{\kern.03em}}{}}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{slovene}
⟨/code⟩
```

## 54 The Russian language

The file `russianb.dtx`[62] defines all the language-specific macros for the Russian language. It needs the file `cyrcod` for success documentation with Russian encodings (see below).

For this language the character `"` is made active. In table 27 an overview is given of its purpose.

| | |
|---|---|
| `"\|` | disable ligature at this position. |
| `"-` | an explicit hyphen sign, allowing hyphenation in the rest of the word. |
| `"---` | Cyrillic emdash in plain text. |
| `"--~` | Cyrillic emdash in compound names (surnames). |
| `"--*` | Cyrillic emdash for denoting direct speech. |
| `""` | like `"-`, but producing no hyphen sign (for compund words with hyphen, e.g. `x-""y` or some other signs as “disable/enable”). |
| `"~` | for a compound word mark without a breakpoint. |
| `"=` | for a compound word mark with a breakpoint, allowing hyphenation in the composing words. |
| `",` | thinspace for initials with a breakpoint in following surname. |
| `"‘` | for German left double quotes (looks like „). |
| `"’` | for German right double quotes (looks like “). |
| ” `"<` | for French left double quotes (looks like ≪). |
| `">` | for French right double quotes (looks like ≫). |

Table 27: The extra definitions made by `russianb`

The quotes in table 27 can also be typeset by using the commands in table 28.

| | |
|---|---|
| `\cdash---` | Cyrillic emdash in plain text. |
| `\cdash--~` | Cyrillic emdash in compound names (surnames). |
| `\cdash--*` | Cyrillic emdash for denoting direct speech. |
| `\glqq` | for German left double quotes (looks like „). |
| `\grqq` | for German right double quotes (looks like “). |
| ” `\flqq` | for French left double quotes (looks like ≪). |
| `\frqq` | for French right double quotes (looks like ≫). |
| `\dq` | the original quotes character (`"`). |

Table 28: More commands which produce quotes, defined by babel

[62]The file described in this section has version number ? and was last revised on ?. This file was initially derived from the original version of `german.sty`, which has some definitions for Russian. Later the definitions from `russian.sty` version 1.0b (for LaTeX 2.09), `russian.sty` version v2.5c (for LaTeX $2_\varepsilon$) and `francais.sty` version 4.5c and `germanb.sty` version 2.5c were added.

The French quotes are also available as ligatures '<<' and '>>' in 8-bit Cyrillic font encodings (`LCY`, `X2`, `T2*`) and as '<' and '>' characters in 7-bit Cyrillic font encodings (`OT2` and `LWN`).

The quotation marks traditionally used in Russian were borrowed from other languages (e.g., French and German) so they keep their original names.

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
⟨*code⟩
\LdfInit{russian}{captionsrussian}
```

When this file is read as an option, i.e., by the `\usepackage` command, `russianb` will be an 'unknown' language, in which case we have to make it known. So we check for the existence of `\l@russian` to see whether we have to do something here.

```
\ifx\l@russian\@undefined
  \@nopatterns{Russian}
  \adddialect\l@russian0
\fi
```

`\latinencoding` We need to know the encoding for text that is supposed to be which is active at the end of the babel package. If the `fontenc` package is loaded later, then... too bad!

```
\let\latinencoding\cf@encoding
```

The user may choose between different available Cyrillic encodings—e.g., `X2`, `LCY`, or `LWN`. Hopefully, `X2` will eventually replace the two latter encodings (`LCY` and `LWN`). If the user wants to use another font encoding than the default (`T2A`), he has to load the corresponding file *before* `russianb.sty`. This may be done in the following way:

```
% override the default X2 encoding used in Babel
\usepackage[LCY,OT1]{fontenc}
\usepackage[english,russian]{babel}
```

Note: for the Russian language, the `T2A` encoding is better than `X2`, because `X2` does not contain Latin letters, and users should be very careful to switch the language every time they want to typeset a Latin word inside a Russian phrase or vice versa.

We parse the `\cdp@list` containing the encodings known to LaTeX in the order they were loaded. We set the `\cyrillicencoding` to the *last* loaded encoding in the list of supported Cyrillic encodings: `OT2`, `LWN`, `LCY`, `X2`, `T2C`, `T2B`, `T2A`, if any.

```
\def\reserved@a#1#2{%
  \edef\reserved@b{#1}%
  \edef\reserved@c{#2}%
  \ifx\reserved@b\reserved@c
    \let\cyrillicencoding\reserved@c
  \fi}
```

```
\def\cdp@elt#1#2#3#4{%
  \reserved@a{#1}{OT2}%
  \reserved@a{#1}{LWN}%
  \reserved@a{#1}{LCY}%
  \reserved@a{#1}{X2}%
  \reserved@a{#1}{T2C}%
  \reserved@a{#1}{T2B}%
  \reserved@a{#1}{T2A}}
\cdp@list
```

Now, if `\cyrillicencoding` is undefined, then the user did not load any of supported encodings. So, we have to set `\cyrillicencoding` to some default value. We test the presence of the encoding definition files in the order from less preferable to more preferable encodings. We use the lowercase names (i.e., `lcyenc.def` instead of `LCYenc.def`).

```
\ifx\cyrillicencoding\undefined
  \IfFileExists{ot2enc.def}{\def\cyrillicencoding{OT2}}\relax
  \IfFileExists{lwnenc.def}{\def\cyrillicencoding{LWN}}\relax
  \IfFileExists{lcyenc.def}{\def\cyrillicencoding{LCY}}\relax
  \IfFileExists{x2enc.def}{\def\cyrillicencoding{X2}}\relax
  \IfFileExists{t2cenc.def}{\def\cyrillicencoding{T2C}}\relax
  \IfFileExists{t2benc.def}{\def\cyrillicencoding{T2B}}\relax
  \IfFileExists{t2aenc.def}{\def\cyrillicencoding{T2A}}\relax
```

If `\cyrillicencoding` is still undefined, then the user seems not to have a properly installed distribution. A fatal error.

```
  \ifx\cyrillicencoding\undefined
    \PackageError{babel}%
      {No Cyrillic encoding definition files were found}%
      {Your installation is incomplete.\MessageBreak
       You need at least one of the following files:\MessageBreak
       \space\space
       x2enc.def, t2aenc.def, t2benc.def, t2cenc.def,\MessageBreak
       \space\space
       lcyenc.def, lwnenc.def, ot2enc.def.}%
  \else
```

We avoid `\usepackage[\cyrillicencoding]{fontenc}` because we don't want to force the switch of `\encodingdefault`.

```
    \lowercase
      \expandafter{\expandafter\input\cyrillicencoding enc.def\relax}%
  \fi
\fi
```

```
    \PackageInfo{babel}
      {Using `\cyrillicencoding' as a default Cyrillic encoding}%
```

```
\DeclareRobustCommand{\Russian}{%
  \fontencoding\cyrillicencoding\selectfont
  \let\encodingdefault\cyrillicencoding
  \expandafter\set@hyphenmins\russianhyphenmins
  \language\l@russian}%
\DeclareRobustCommand{\English}{%
  \fontencoding\latinencoding\selectfont
  \let\encodingdefault\latinencoding
  \expandafter\set@hyphenmins\englishhyphenmins
  \language\l@english}%
\let\Rus\Russian
\let\Eng\English
\let\cyrillictext\Russian
\let\cyr\Russian
```

Since the `X2` encoding does not contain Latin letters, we should make some redefinitions of LaTeX macros which implicitly produce Latin letters.

```
\expandafter\ifx\csname T@X2\endcsname\relax\else
```

We put `\latinencoding` in braces to avoid problems with `\@alph` inside minipages (e.g., footnotes inside minipages) where `\@alph` is expanded and we get for example '`\fontencoding OT1`' (`\fontencoding` is robust).

```
  \def\@alph#1{{\fontencoding{\latinencoding}\selectfont
    \ifcase#1\or
      a\or b\or c\or d\or e\or f\or g\or h\or
      i\or j\or k\or l\or m\or n\or o\or p\or
      q\or r\or s\or t\or u\or v\or w\or x\or
      y\or z\else\@ctrerr\fi}}%
  \def\@Alph#1{{\fontencoding{\latinencoding}\selectfont
    \ifcase#1\or
      A\or B\or C\or D\or E\or F\or G\or H\or
      I\or J\or K\or L\or M\or N\or O\or P\or
      Q\or R\or S\or T\or U\or V\or W\or X\or
      Y\or Z\else\@ctrerr\fi}}%
```

Unfortunately, the commands `\AA` and `\aa` are not encoding dependent in LaTeX (unlike e.g., `\oe` or `\DH`). They are defined as `\r{A}` and `\r{a}`. This leads to unpredictable results when the font encoding does not contain the Latin letters 'A' and 'a' (like `X2`).

```
  \DeclareTextSymbolDefault{\AA}{OT1}
  \DeclareTextSymbolDefault{\aa}{OT1}
  \DeclareTextCommand{\aa}{OT1}{\r a}
  \DeclareTextCommand{\AA}{OT1}{\r A}
\fi
```

The following block redefines the character class of uppercase Greek letters and some accents, if it is equal to 7 (variable family), to avoid incorrect results if the font encoding in some math family does not contain these characters in places of OT1 encoding. The code was taken from `amsmath.dtx`. See comments and further explanation there.

```
% \begingroup\catcode`\"=12
% % uppercase greek letters:
% \def\@tempa#1{\expandafter\@tempb\meaning#1\relax\relax\relax\relax
%   "0000\@nil#1}
% \def\@tempb#1"#2#3#4#5#6\@nil#7{%
%   \ifnum"#2=7 \count@"1#3#4#5\relax
%     \ifnum\count@<"1000 \else \global\mathchardef#7="0#3#4#5\relax \fi
%   \fi}
% \@tempa\Gamma\@tempa\Delta\@tempa\Theta\@tempa\Lambda\@tempa\Xi
% \@tempa\Pi\@tempa\Sigma\@tempa\Upsilon\@tempa\Phi\@tempa\Psi
% \@tempa\Omega
% % some accents:
% \def\@tempa#1#2\@nil{\def\@tempc{#1}}\def\@tempb{\mathaccent}
% \expandafter\@tempa\hat\relax\relax\@nil
% \ifx\@tempb\@tempc
%   \def\@tempa#1\@nil{#1}%
%   \def\@tempb#1{\afterassignment\@tempa\mathchardef\@tempc=}%
%   \def\do#1"#2{}
%   \def\@tempd#1{\expandafter\@tempb#1\@nil
%     \ifnum\@tempc>"FFF
%       \xdef#1{\mathaccent"\expandafter\do\meaning\@tempc\space}%
%     \fi}
%   \@tempd\hat\@tempd\check\@tempd\tilde\@tempd\acute\@tempd\grave
%   \@tempd\dot\@tempd\ddot\@tempd\breve\@tempd\bar
% \fi
% \endgroup
```

The user should use the `inputenc` package when any 8-bit Cyrillic font encoding is used, selecting one of the Cyrillic input encodings. We do not assume any default input encoding, so the user should explicitly call the `inputenc` package by `\usepackage{inputenc}`. We also removed `\AtBeginDocument`, so `inputenc` should be used before babel.

```
\@ifpackageloaded{inputenc}{}{%
  \def\reserved@a{LWN}%
  \ifx\reserved@a\cyrillicencoding\else
    \def\reserved@a{OT2}%
    \ifx\reserved@a\cyrillicencoding\else
      \PackageWarning{babel}%
        {No input encoding specified for Russian language}
  \fi\fi}
```

Now we define two commands that offer the possibility to switch between Cyrillic and Roman encodings.

`\cyrillictext` `\latintext` The command `\cyrillictext` will switch from Latin font encoding to the Cyrillic font encoding, the command `\latintext` switches back. This assumes that the 'normal' font encoding is a Latin one. These commands are *declarations*, for shorter peaces of text the commands `\textlatin` and `\textcyrillic` can be used.

```
%\DeclareRobustCommand{\latintext}{%
%  \fontencoding{\latinencoding}\selectfont
%  \def\encodingdefault{\latinencoding}}
\let\lat\latintext
```

\textcyrillic \textlatin These commands take an argument which is then typeset using the requested font encoding.

```
\DeclareTextFontCommand{\textcyrillic}{\cyrillictext}
%\DeclareTextFontCommand{\textlatin}{\latintext}
```

We make the TeX

```
%\ifx\ltxTeX\undefined\let\ltxTeX\TeX\fi
%\ProvideTextCommandDefault{\TeX}{\textlatin{\ltxTeX}}
```

and LaTeX logos encoding independent.

```
%\ifx\ltxLaTeX\undefined\let\ltxLaTeX\LaTeX\fi
%\ProvideTextCommandDefault{\LaTeX}{\textlatin{\ltxLaTeX}}
```

The next step consists of defining commands to switch to (and from) the Russian language.

\captionsrussian The macro `\captionsrussian` defines all strings used in the four standard document classes provided with LaTeX. The two commands `\cyr` and `\lat` activate Cyrillic resp. Latin encoding.

```
\addto\captionsrussian{%
%   FIXME: Where is the \prefacename used?
  \def\prefacename{%
    {\cyr\CYRP\cyrr\cyre\cyrd\cyri\cyrs\cyrl\cyro\cyrv\cyri\cyre}}%
%   {\cyr\CYRV\cyrv\cyre\cyrd\cyre\cyrn\cyri\cyre}}%
  \def\refname{%
    {\cyr\CYRS\cyrp\cyri\cyrs\cyro\cyrk
      \ \cyrl\cyri\cyrt\cyre\cyrr\cyra\cyrt\cyru\cyrr\cyrery}}%
% \def\refname{%
%   {\cyr\CYRL\cyri\cyrt\cyre\cyrr\cyra\cyrt\cyru\cyrr\cyra}}%
  \def\abstractname{%
    {\cyr\CYRA\cyrn\cyrn\cyro\cyrt\cyra\cyrc\cyri\cyrya}}%
  \def\bibname{%
    {\cyr\CYRL\cyri\cyrt\cyre\cyrr\cyra\cyrt\cyru\cyrr\cyra}}%
% \def\bibname{%
%   {\cyr\CYRB\cyri\cyrb\cyrl\cyri\cyro
%    \cyrg\cyrr\cyra\cyrf\cyri\cyrya}}%
% for reports according to GOST:
% \def\bibname{%
%   {\cyr\CYRS\cyrp\cyri\cyrs\cyro\cyrk
%     \ \cyri\cyrs\cyrp\cyro\cyrl\cyrsftsn\cyrz\cyro\cyrv\cyra\cyrn
%     \cyrn\cyrery\cyrh\ \cyri\cyrs\cyrt\cyro\cyrch\cyrn\cyri
%     \cyrk\cyro\cyrv}}%
  \def\chaptername{{\cyr\CYRG\cyrl\cyra\cyrv\cyra}}%
% \@ifundefined{chapter}{}{%
%   \def\chaptername{{\cyr\CYRG\cyrl\cyra\cyrv\cyra}}}%
  \def\appendixname{%
    {\cyr\CYRP\cyrr\cyri\cyrl\cyro\cyrzh\cyre\cyrn\cyri\cyre}}%
```

There are two names for the Table of Contents that are used in Russian publications. For books (and reports) the second variant is appropriate, but for proceedings the first variant is preferred:

```
  \@ifundefined{thechapter}%
    {\def\contentsname{%
      {\cyr\CYRS\cyro\cyrd\cyre\cyrr\cyrzh\cyra\cyrn\cyri\cyre}}}%
    {\def\contentsname{%
      {\cyr\CYRO\cyrg\cyrl\cyra\cyrv\cyrl\cyre\cyrn\cyri\cyre}}}%
  \def\listfigurename{%
    {\cyr\CYRS\cyrp\cyri\cyrs\cyro\cyrk
      \ \cyri\cyrl\cyrl\cyryu\cyrs\cyrt\cyrr\cyra\cyrc\cyri\cyrishrt}}%
% \def\listfigurename{%
%   {\cyr\CYRS\cyrp\cyri\cyrs\cyro\cyrk
%     \ \cyrr\cyri\cyrs\cyru\cyrn\cyrk\cyro\cyrv}}%
  \def\listtablename{%
    {\cyr\CYRS\cyrp\cyri\cyrs\cyro\cyrk
      \ \cyrt\cyra\cyrb\cyrl\cyri\cyrc}}%
  \def\indexname{%
    {\cyr\CYRP\cyrr\cyre\cyrd\cyrm\cyre\cyrt\cyrn\cyrery\cyrishrt
      \ \cyru\cyrk\cyra\cyrz\cyra\cyrt\cyre\cyrl\cyrsftsn}}%
  \def\authorname{%
    {\cyr\CYRI\cyrm\cyre\cyrn\cyrn\cyro\cyrishrt
      \ \cyru\cyrk\cyra\cyrz\cyra\cyrt\cyre\cyrl\cyrsftsn}}%
  \def\figurename{{\cyr\CYRR\cyri\cyrs.}}%
  \def\tablename{{\cyr\CYRT\cyra\cyrb\cyrl\cyri\cyrc\cyra}}%
  \def\partname{{\cyr\CYRCH\cyra\cyrs\cyrt\cyrsftsn}}%
  \def\enclname{{\cyr\cyrv\cyrk\cyrl.}}%
  \def\ccname{{\cyr\cyri\cyrs\cyrh.}}%
% \def\ccname{{\cyr\cyri\cyrz}}%
  \def\headtoname{{\cyr\cyrv\cyrh.}}%
% \def\headtoname{{\cyr\cyrv}}%
  \def\pagename{{\cyr\cyrs.}}%
% \def\pagename{{\cyr\cyrs\cyrt\cyrr.}}%
  \def\seename{{\cyr\cyrs\cyrm.}}%
  \def\alsoname{{\cyr\cyrs\cyrm.\ \cyrt\cyra\cyrk\cyrzh\cyre}}%

  \def\proofname{{\cyr\CYRD\cyro\cyrk\cyra\cyrz\cyra\cyrt
      \cyre\cyrl\cyrsftsn\cyrs\cyrt\cyrv\cyro}}%
  \def\glossaryname{Glossary}% <-- Needs translation
  }
```

`\daterussian` The macro `\daterussian` redefines the command `\today` to produce Russian dates.

```
\def\daterussian{%
  \def\today{\number\day~\ifcase\month\or
    \cyrya\cyrn\cyrv\cyra\cyrr\cyrya\or
    \cyrf\cyre\cyrv\cyrr\cyra\cyrl\cyrya\or
    \cyrm\cyra\cyrr\cyrt\cyra\or
    \cyra\cyrp\cyrr\cyre\cyrl\cyrya\or
\cyrm\cyra\cyrya\or
\cyri\cyryu\cyrn\cyrya\or
\cyri\cyryu\cyrl\cyrya\or
\cyra\cyrv\cyrg\cyru\cyrs\cyrt\cyra\or
\cyrs\cyre\cyrn\cyrt\cyrya\cyrb\cyrr\cyrya\or
\cyro\cyrk\cyrt\cyrya\cyrb\cyrr\cyrya\or
\cyrn\cyro\cyrya\cyrb\cyrr\cyrya\or
\cyrd\cyre\cyrk\cyra\cyrb\cyrr\cyrya\fi
\ \number\year~\cyrg.}}
```

`\extrasrussian` The macro `\extrasrussian` will perform all the extra definitions needed for the Russian language. The macro `\noextrasrussian` is used to cancel the actions of `\extrasrussian`.

The first action we define is to switch on the selected Cyrillic encoding whenever we enter 'russian'.

```
\addto\extrasrussian{\cyrillictext}
```

When the encoding definition file was processed by LaTeX the current font encoding is stored in `\latinencoding`, assuming that LaTeX uses `T1` or `OT1` as default. Therefore we switch back to `\latinencoding` whenever the Russian language is no longer 'active'.

```
\addto\noextrasrussian{\latintext}
```

`\verbatim@font` In order to get both Latin and Cyrillic letters in verbatim text we need to change the definition of an internal LaTeX command somewhat:

```
%\def\verbatim@font{%
%  \let\encodingdefault\latinencoding
%  \normalfont\ttfamily
%  \expandafter\def\csname\cyrillicencoding-cmd\endcsname##1##2{%
%    \ifx\protect\@typeset@protect
%      \begingroup\UseTextSymbol\cyrillicencoding##1\endgroup
%    \else\noexpand##1\fi}}
```

The category code of the characters '`:`', '`;`', '`!`', and '`?`' is made `\active` to insert a little white space.

For Russian (as well as for German) the `"` character also is made active.

Note: It is *very* questionable whether the Russian typesetting tradition requires additional spacing before those punctuation signs. Therefore, we make the corresponding code optional. If you need it, then define the `frenchpunct` docstrip option in `babel.ins`.

Borrowed from french. Some users dislike automatic insertion of a space before 'double punctuation', and prefer to decide themselves whether a space should be added or not; so a hook `\NoAutoSpaceBeforeFDP` is provided: if this command is added (in file `russianb.cfg`, or anywhere in a document) `russianb` will respect your typing, and introduce a suitable space before 'double punctuation' *if and only if* a space is typed in the source file before those signs.

The command `\AutoSpaceBeforeFDP` switches back to the default behavior of `russianb`.

```
⟨*frenchpunct⟩
\initiate@active@char{:}
\initiate@active@char{;}
⟨/frenchpunct⟩
⟨*frenchpunct | spanishligs⟩
\initiate@active@char{!}
\initiate@active@char{?}
⟨/frenchpunct | spanishligs⟩
\initiate@active@char{"}
```

The code above is necessary because we need extra active characters. The character `"` is used as indicated in table 27.

We specify that the Russian group of shorthands should be used.

```
\addto\extrasrussian{\languageshorthands{russian}}
```

These characters are 'turned on' once, later their definition may vary.

```
\addto\extrasrussian{%
⟨frenchpunct⟩   \bbl@activate{:}\bbl@activate{;}%
⟨frenchpunct | spanishligs⟩   \bbl@activate{!}\bbl@activate{?}%
  \bbl@activate{"}}
\addto\noextrasrussian{%
⟨frenchpunct⟩   \bbl@deactivate{:}\bbl@deactivate{;}%
⟨frenchpunct | spanishligs⟩   \bbl@deactivate{!}\bbl@deactivate{?}%
  \bbl@deactivate{"}}
```

The `X2` and `T2*` encodings do not contain `spanish_shriek` and `spanish_query` symbols; as a consequence, the ligatures '`?‘`' and '`!‘`' do not work with them (these characters are useless for Cyrillic texts anyway). But we define the shorthands to emulate these ligatures (optionally).

We do not use `\latinencoding` here (but instead explicitly use `OT1`) because the user may choose `T2A` to be the primary encoding, but it does not contain these characters.

```
⟨*spanishligs⟩
\declare@shorthand{russian}{?‘}{\UseTextSymbol{OT1}\textquestiondown}
\declare@shorthand{russian}{!‘}{\UseTextSymbol{OT1}\textexclamdown}
⟨/spanishligs⟩
```

`\russian@sh@;@`
`\russian@sh@:@`
`\russian@sh@!@`
`\russian@sh@?@`

We have to reduce the amount of white space before `;`, `:` and `!`. This should only happen in horizontal mode, hence the test with `\ifhmode`.

```
⟨*frenchpunct⟩
\declare@shorthand{russian}{;}{%
  \ifhmode
```

In horizontal mode we check for the presence of a 'space', 'unskip' if it exists and place a `0.1em` kerning.

```
    \ifdim\lastskip>\z@
      \unskip\nobreak\kern.1em
    \else
```

If no space has been typed, we add \FDP@thinspace which will be defined, up to the user's wishes, as an automatic added thinspace, or as \@empty.

```
          \FDP@thinspace
    \fi
  \fi
```

Now we can insert a '`;`' character.

```
  \string;}
```

The other definitions are very similar.

```
\declare@shorthand{russian}{:}{%
  \ifhmode
    \ifdim\lastskip>\z@
      \unskip\nobreak\kern.1em
    \else
        \FDP@thinspace
    \fi
  \fi
  \string:}
\declare@shorthand{russian}{!}{%
  \ifhmode
    \ifdim\lastskip>\z@
      \unskip\nobreak\kern.1em
    \else
        \FDP@thinspace
    \fi
  \fi
  \string!}
\declare@shorthand{russian}{?}{%
  \ifhmode
    \ifdim\lastskip>\z@
      \unskip\nobreak\kern.1em
    \else
        \FDP@thinspace
    \fi
  \fi
  \string?}
```

\AutoSpaceBeforeFDP
\NoAutoSpaceBeforeFDP
\FDP@thinspace

\FDP@thinspace is defined as unbreakable spaces if \AutoSpaceBeforeFDP is activated or as \@empty if \NoAutoSpaceBeforeFDP is in use. The default is \AutoSpaceBeforeFDP.

```
\def\AutoSpaceBeforeFDP{%
      \def\FDP@thinspace{\nobreak\kern.1em}}
\def\NoAutoSpaceBeforeFDP{\let\FDP@thinspace\@empty}
\AutoSpaceBeforeFDP
```

\FDPon
\FDPoff

The next macros allow to switch on/off activeness of double punctuation signs.

```
\def\FDPon{\bbl@activate{:}%
        \bbl@activate{;}%
        \bbl@activate{?}%
        \bbl@activate{!}}
\def\FDPoff{\bbl@deactivate{:}%
        \bbl@deactivate{;}%
        \bbl@deactivate{?}%
        \bbl@deactivate{!}}
```

\system@sh@:@
\system@sh@!@
\system@sh@?@
\system@sh@;@

When the active characters appear in an environment where their Russian behaviour is not wanted they should give an 'expected' result. Therefore we define shorthands at system level as well.

```
\declare@shorthand{system}{:}{\string:}
\declare@shorthand{system}{;}{\string;}
⟨/frenchpunct⟩
⟨*frenchpunct&!spanishligs⟩
\declare@shorthand{system}{!}{\string!}
\declare@shorthand{system}{?}{\string?}
⟨/frenchpunct&!spanishligs⟩
```

To be able to define the function of '"', we first define a couple of 'support' macros.

\dq

We save the original double quote character in `\dq` to keep it available, the math accent `\"` can now be typed as '"'.

```
\begingroup \catcode`\"12
\def\reserved@a{\endgroup
  \def\@SS{\mathchar"7019 }
  \def\dq{"}}
\reserved@a
```

Now we can define the doublequote macros: german and french quotes. We use definitions of these quotes made in babel.sty. The french quotes are contained in the `T2*` encodings.

```
\declare@shorthand{russian}{"`}{\glqq}
\declare@shorthand{russian}{"'}{\grqq}
\declare@shorthand{russian}{"<}{\flqq}
\declare@shorthand{russian}{">}{\frqq}
```

Some additional commands:

```
\declare@shorthand{russian}{""}{\hskip\z@skip}
\declare@shorthand{russian}{"~}{\textormath{\leavevmode\hbox{-}}{-}}
\declare@shorthand{russian}{"=}{\nobreak-\hskip\z@skip}
\declare@shorthand{russian}{"|}{%
  \textormath{\nobreak\discretionary{-}{}{\kern.03em}%
              \allowhyphens}{}}
```

The next two macros for `"-` and `"---` are somewhat different. We must check whether the second token is a hyphen character:

```
\declare@shorthand{russian}{"-}{%
```

If the next token is '-', we typeset an emdash, otherwise a hyphen sign:

```
  \def\russian@sh@tmp{%
    \if\russian@sh@next-\expandafter\russian@sh@emdash
    \else\expandafter\russian@sh@hyphen\fi
  }%
```

TEX looks for the next token after the first '-': the meaning of this token is written to `\russian@sh@next` and `\russian@sh@tmp` is called.

```
  \futurelet\russian@sh@next\russian@sh@tmp}
```

Here are the definitions of hyphen and emdash. First the hyphen:

```
\def\russian@sh@hyphen{%
  \nobreak\-\bbl@allowhyphens}
```

For the emdash definition, there are the two parameters: we must 'eat' two last hyphen signs of our emdash. . . :

```
\def\russian@sh@emdash#1#2{\cdash-#1#2}
```

`\cdash` . . . these two parameters are useful for another macro: `\cdash`:

```
%\ifx\cdash\undefined % should be defined earlier
\def\cdash#1#2#3{\def\tempx@{#3}%
\def\tempa@{-}\def\tempb@{~}\def\tempc@{*}%
 \ifx\tempx@\tempa@\@Acdash\else
  \ifx\tempx@\tempb@\@Bcdash\else
   \ifx\tempx@\tempc@\@Ccdash\else
    \errmessage{Wrong usage of cdash}\fi\fi\fi}
```

second parameter (or third for `\cdash`) shows what kind of emdash to create in next step

`"---` ordinary (plain) Cyrillic emdash inside text: an unbreakable thinspace will be inserted before only in case of a *space* before the dash (it is necessary for dashes after display maths formulae: there could be lists, enumerations etc. started with "— where *a* is ..." i.e., the dash starts a line). (Firstly there were planned rather soft rules for user: he may put a space before the dash or not. But it is difficult to place this thinspace automatically, i.e., by checking modes because after display formulae TEX uses horizontal mode. Maybe there is a misunderstanding? Maybe there is another way?) After a dash a breakable thinspace is always placed;

```
% What is more grammatically: .2em or .2\fontdimen6\font ?
\def\@Acdash{\ifdim\lastskip>\z@\unskip\nobreak\hskip.2em\fi
  \cyrdash\hskip.2em\ignorespaces}%
```

`"--~` emdash in compound names or surnames (like Mendeleev–Klapeiron); this dash has no space characters around; after the dash some space is added `\exhyphenalty`

```
\def\@Bcdash{\leavevmode\ifdim\lastskip>\z@\unskip\fi
 \nobreak\cyrdash\penalty\exhyphenpenalty\hskip\z@skip\ignorespaces}%
```

`"--*` for denoting direct speech (a space like `\enskip` must follow the emdash);

```
\def\@Ccdash{\leavevmode
 \nobreak\cyrdash\nobreak\hskip.35em\ignorespaces}%
%\fi
```

`\cyrdash` Finally the macro for "body" of the Cyrillic emdash. The `\cyrdash` macro will be defined in case this macro hasn't been defined in a fontenc file. For T2* fonts, cyrdash will be placed in the code of the English emdash thus it uses ligature `---`.

```
% Is there an IF necessary?
\ifx\cyrdash\undefined
  \def\cyrdash{\hbox to.8em{--\hss--}}
\fi
```

Here a really new macro—to place thinspace between initials. This macro used instead of `\,` allows hyphenation in the following surname.

```
\declare@shorthand{russian}{",}{\nobreak\hskip.2em\ignorespaces}
```

`\mdqon` `\mdqoff` All that's left to do now is to define a couple of commands for `"`.

```
\def\mdqon{\bbl@activate{"}}
\def\mdqoff{\bbl@deactivate{"}}
```

The Russian hyphenation patterns can be used with `\lefthyphenmin` and `\righthyphenmin` set to 2.

```
\providehyphenmins{\CurrentOption}{\tw@\tw@}
% temporary hack:
\ifx\englishhyphenmins\undefined
  \def\englishhyphenmins{\tw@\thr@@}
\fi
```

Now the action `\extrasrussian` has to execute is to make sure that the command `\frenchspacing` is in effect. If this is not the case the execution of `\noextrasrussian` will switch it off again.

```
\addto\extrasrussian{\bbl@frenchspacing}
\addto\noextrasrussian{\bbl@nonfrenchspacing}
```

Next we add a new enumeration style for Russian manuscripts with Cyrillic letters, and later on we define some math operator names in accordance with Russian typesetting traditions.

`\Asbuk` We begin by defining `\Asbuk` which works like `\Alph`, but produces (uppercase) Cyrillic letters intead of Latin ones. The letters YO, ISHRT, HRDSN, ERY, and SFTSN are skipped, as usual for such enumeration.

```
\def\Asbuk#1{\expandafter\@Asbuk\csname c@#1\endcsname}
\def\@Asbuk#1{\ifcase#1\or
  \CYRA\or\CYRB\or\CYRV\or\CYRG\or\CYRD\or\CYRE\or\CYRZH\or
  \CYRZ\or\CYRI\or\CYRK\or\CYRL\or\CYRM\or\CYRN\or\CYRO\or
  \CYRP\or\CYRR\or\CYRS\or\CYRT\or\CYRU\or\CYRF\or\CYRH\or
  \CYRC\or\CYRCH\or\CYRSH\or\CYRSHCH\or\CYREREV\or\CYRYU\or
  \CYRYA\else\@ctrerr\fi}
```

`\asbuk` The macro `\asbuk` is similar to `\alph`; it produces lowercase Russian letters.

```
\def\asbuk#1{\expandafter\@asbuk\csname c@#1\endcsname}
\def\@asbuk#1{\ifcase#1\or
  \cyra\or\cyrb\or\cyrv\or\cyrg\or\cyrd\or\cyre\or\cyrzh\or
  \cyrz\or\cyri\or\cyrk\or\cyrl\or\cyrm\or\cyrn\or\cyro\or
  \cyrp\or\cyrr\or\cyrs\or\cyrt\or\cyru\or\cyrf\or\cyrh\or
  \cyrc\or\cyrch\or\cyrsh\or\cyrshch\or\cyrerev\or\cyryu\or
  \cyrya\else\@ctrerr\fi}
```

Set up default Cyrillic math alphabets. To use Cyrillic letters in math mode user should load the `textmath` package *before* loading fontenc package (or babel). Note, that by default Cyrillic letters are taken from upright font in math mode (unlike Latin letters).

```
%\RequirePackage{textmath}
\@ifundefined{sym\cyrillicencoding letters}{}{%
\SetSymbolFont{\cyrillicencoding letters}{bold}\cyrillicencoding
  \rmdefault\bfdefault\updefault
\DeclareSymbolFontAlphabet\cyrmathrm{\cyrillicencoding letters}
```

And we need a few commands to be able to switch to different variants.

```
\DeclareMathAlphabet\cyrmathbf\cyrillicencoding
  \rmdefault\bfdefault\updefault
\DeclareMathAlphabet\cyrmathsf\cyrillicencoding
  \sfdefault\mddefault\updefault
\DeclareMathAlphabet\cyrmathit\cyrillicencoding
  \rmdefault\mddefault\itdefault
\DeclareMathAlphabet\cyrmathtt\cyrillicencoding
  \ttdefault\mddefault\updefault
%
\SetMathAlphabet\cyrmathsf{bold}\cyrillicencoding
  \sfdefault\bfdefault\updefault
\SetMathAlphabet\cyrmathit{bold}\cyrillicencoding
  \rmdefault\bfdefault\itdefault
}
```

Some math functions in Russian math books have other names: e.g., `sinh` in Russian is written as `sh` etc. So we define a number of new math operators.

`\sinh`:

```
\def\sh{\mathop{\operator@font sh}\nolimits}
```

`\cosh`:

```
\def\ch{\mathop{\operator@font ch}\nolimits}
```

`\tan`:

```
\def\tg{\mathop{\operator@font tg}\nolimits}
```

`\arctan`:

```
\def\arctg{\mathop{\operator@font arctg}\nolimits}
```

arcctg:

```
\def\arcctg{\mathop{\operator@font arcctg}\nolimits}
```

The following macro conflicts with `\th` defined in Latin 1 encoding:

`\tanh`:

```
\addto\extrasrussian{%
  \babel@save{\th}%
  \let\ltx@th\th
  \def\th{\textormath{\ltx@th}%
                     {\mathop{\operator@font th}\nolimits}}%
  }
```

`\cot`:

```
\def\ctg{\mathop{\operator@font ctg}\nolimits}
```

`\coth`:

```
\def\cth{\mathop{\operator@font cth}\nolimits}
```

`\csc`:

```
\def\cosec{\mathop{\operator@font cosec}\nolimits}
```

And finally some other Russian mathematical symbols:

```
\def\Prob{\mathop{\kern\z@\mathsf{P}}\nolimits}
\def\Variance{\mathop{\kern\z@\mathsf{D}}\nolimits}
\def\nod{\mathop{\cyrmathrm{\cyrn.\cyro.\cyrd.}}\nolimits}
\def\nok{\mathop{\cyrmathrm{\cyrn.\cyro.\cyrk.}}\nolimits}
\def\NOD{\mathop{\cyrmathrm{\CYRN\CYRO\CYRD}}\nolimits}
\def\NOK{\mathop{\cyrmathrm{\CYRN\CYRO\CYRK}}\nolimits}
\def\Proj{\mathop{\cyrmathrm{\CYRP\cyrr}}\nolimits}
```

This is for compatibility with older Russian packages.

```
\DeclareRobustCommand{\No}{%
   \ifmmode{\nfss@text{\textnumero}}\else\textnumero\fi}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{russian}
⟨/code⟩
```

## 55 The Bulgarian language

The file `bulgarian.dtx`[63] provides the language-specific macros for the Bulgarian language.

Users should take note of the vaious "cyrillic" dashes available now (see below). These should remove many causes of headache. Also, although by default the Bulgarian quotation marks will appear automatically when typesetting in Bulgarian, it is better to use the new commands `\"'` and `\"'` which explicitly typeset them. Note: automatic switch to Bulgarian quotation is withdrawn for the moment and may not be reintroduced at all.

For this language the character `"` is made active. In table 29 an overview is given of its purpose.

| | |
|---|---|
| `"\|` | disable ligature at this position. |
| `"-` | an explicit hyphen sign, allowing hyphenation in the rest of the word. |
| `"---` | Cyrillic emdash in plain text. |
| `"--~` | Cyrillic emdash in compound names (surnames). |
| `"--*` | Cyrillic emdash for denoting direct speech. |
| `""` | like `"-`, but producing no hyphen sign (for compound words with hyphen, e.g. `x-""y` or some other signs as "disable/enable"). |
| `"~` | for a compound word mark without a breakpoint. |
| `"=` | for a compound word mark with a breakpoint, allowing hyphenation in the composing words. |
| `",` | thinspace for initials with a breakpoint in following surname. |
| `"‘` | for German left double quotes (looks like „). |
| `"’` | for German right double quotes (looks like “). |
| ” `"<` | for French left double quotes (looks like ≪). |
| `">` | for French right double quotes (looks like ≫). |

Table 29: The extra definitions made by`bulgarian`

The quotes in table 29 can also be typeset by using the commands in table 30.

The French quotes are also available as ligatures ‘`<<`’ and ‘`>>`’ in 8-bit Cyrillic font encodings (`LCY`, `X2`, `T2*`) and as ‘`<`’ and ‘`>`’ characters in 7-bit Cyrillic font encodings (`OT2` and `LWN`).

The quotation marks traditionally used in Bulgarian were borrowed from German o they keep their original names. French quotation marks may be seen as well in older books.

[63]The file described in this section has version number ? and was last revised on ?. This file was initially derived from the August-1998 version of `russianb.dtx`.
It is (reasonably) backward compatible with the 1994/1996 (non-babel) bulgarian style (bulgaria.sty) by Georgi Boshnakov—files prepared for that style should compile successfully (with vastly improved appearance due to usage of standard fonts).

| | |
|---|---|
| `\cdash---` | Cyrillic emdash in plain text. |
| `\cdash--~` | Cyrillic emdash in compound names (surnames). |
| `\cdash--*` | Cyrillic emdash for denoting direct speech. |
| `\glqq` | for German left double quotes (looks like„). |
| `\grqq` | for German right double quotes (looks like “). |
| ” `\flqq` | for French left double quotes (looks like ≪). |
| `\frqq` | for French right double quotes (looks like ≫). |
| `\dq` | the original quotes character ("). |

Table 30: More commands which produce quotes, defined by babel

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
55.1 ⟨*code⟩
55.2 \LdfInit{bulgarian}{captionsbulgarian}
```

When this file is read as an option, i.e., by the `\usepackage` command, `bulgarian` will be an ‘unknown’ language, in which case we have to make it known. So we check for the existence of `\l@bulgarian` to see whether we have to do something here.

```
55.3 \ifx\l@bulgarian\@undefined
55.4   \@nopatterns{Bulgarian}
55.5   \adddialect\l@bulgarian0
55.6 \fi
```

`\latinencoding` We need to know the encoding for text that is supposed to be which is active at the end of the babel package. If the `fontenc` package is loaded later, then . . . too bad!

```
55.7 \let\latinencoding\cf@encoding
```

The user may choose between different available Cyrillic encodings—e.g., `X2`, `LCY`, or `LWN`. If the user wants to use a font encoding other than the default (`T2A`), he has to load the corresponding file *before* `bulgarian.sty`. This may be done in the following way:

```
\usepackage[LCY,OT1]{fontenc}     %overwrite the default encoding;
\usepackage[english,bulgarian]{babel}
```

Note: most people would prefer the `T2A` to `X2`, because `X2` does not contain Latin letters, and users should be very careful to switch the language every time they want to typeset a Latin word inside a Bulgarian phrase or vice versa. On the other hand, switching the language is a good practice anyway. With a decent text processing program it does not involve more work than switching between the Bulgarian and English keyboard. Moreover that the far most common disruption occurs as a result of forgetting to switch back to cyrillic keyboard.

We parse the `\cdp@list` containing the encodings known to LaTeX in the order they were loaded. We set the `\cyrillicencoding` to the *last* loaded encoding in the list of supported Cyrillic encodings: `OT2`, `LWN`, `LCY`, `X2`, `T2C`, `T2B`, `T2A`, if any.

```
\def\reserved@a#1#2{%
   \edef\reserved@b{#1}%
   \edef\reserved@c{#2}%
   \ifx\reserved@b\reserved@c
     \let\cyrillicencoding\reserved@c
   \fi}
\def\cdp@elt#1#2#3#4{%
   \reserved@a{#1}{OT2}%
   \reserved@a{#1}{LWN}%
   \reserved@a{#1}{LCY}%
   \reserved@a{#1}{X2}%
   \reserved@a{#1}{T2C}%
   \reserved@a{#1}{T2B}%
   \reserved@a{#1}{T2A}}
\cdp@list
```

Now, if `\cyrillicencoding` is undefined, then the user did not load any of supported encodings. So, we have to set `\cyrillicencoding` to some default value. We test the presence of the encoding definition files in the order from less preferable to more preferable encodings. We use the lowercase names (i.e., `lcyenc.def` instead of `LCYenc.def`).

```
\ifx\cyrillicencoding\undefined
  \IfFileExists{ot2enc.def}{\def\cyrillicencoding{OT2}}\relax
  \IfFileExists{lwnenc.def}{\def\cyrillicencoding{LWN}}\relax
  \IfFileExists{lcyenc.def}{\def\cyrillicencoding{LCY}}\relax
  \IfFileExists{x2enc.def}{\def\cyrillicencoding{X2}}\relax
  \IfFileExists{t2cenc.def}{\def\cyrillicencoding{T2C}}\relax
  \IfFileExists{t2benc.def}{\def\cyrillicencoding{T2B}}\relax
  \IfFileExists{t2aenc.def}{\def\cyrillicencoding{T2A}}\relax
```

If `\cyrillicencoding` is still undefined, then the user seems not to have a properly installed distribution. A fatal error.

```
\ifx\cyrillicencoding\undefined
    \PackageError{babel}%
    {No Cyrillic encoding definition files were found}%
    {Your installation is incomplete. \MessageBreak
    You need at least one of the following files: \MessageBreak
    \space\space
    x2enc.def, t2aenc.def, t2benc.def, t2cenc.def, \MessageBreak
    \space\space
    lcyenc.def, lwnenc.def, ot2enc.def.}%
  \else
```

We avoid `\usepackage[\cyrillicencoding]{fontenc}` because we don't want to force the switch of `\encodingdefault`.

```
    \lowercase
```

```
      \expandafter{\expandafter\input\cyrillicencoding enc.def\relax}%
  \fi
\fi
```

```
    \PackageInfo{babel}
      {Using `\cyrillicencoding' as a default Cyrillic encoding}%
```

```
\DeclareRobustCommand{\Bulgarian}{%
  \fontencoding\cyrillicencoding\selectfont
  \let\encodingdefault\cyrillicencoding
  \expandafter\set@hyphenmins\bulgarianhyphenmins
  \language\l@bulgarian}
\DeclareRobustCommand{\English}{%
  \fontencoding\latinencoding\selectfont
  \let\encodingdefault\latinencoding
  \expandafter\set@hyphenmins\englishhyphenmins
  \language\l@english}
\let\Bul\Bulgarian
\let\Bg\Bulgarian
\let\cyrillictext\Bulgarian
\let\cyr\Bulgarian
\let\Eng\English
\def\selectenglanguage{\selectlanguage{english}}
\def\selectbglanguage{\selectlanguage{bulgarian}}
```

Since the `X2` encoding does not contain Latin letters,we should make some redefinitions of LaTeX macros which implicitly produce Latin letters.

```
\expandafter\ifx\csname T@X2\endcsname\relax\else
```

We put `\latinencoding` in braces to avoid problems with `\@alph` inside minipages (e.g., footnotes inside minipages) where `\@alph` is expanded and we get for example '`\fontencoding OT1`' (`\fontencoding` is robust).

```
  \def\@Alph@eng#1{{\fontencoding{\latinencoding}\selectfont
      \ifcase#1\or A\or B\or C\or D\or E\or F\or G\or H\or I\or J\or
      K\or L\or M\or N\or O\or P\or Q\or R\or S\or T\or U\or V\or W\or
      X\or Y\or Z\else \@ctrerr\fi}}%
  \def\@alph@eng#1{{\fontencoding{\latinencoding}\selectfont
      \ifcase#1\or a\or b\or c\or d\or e\or f\or g\or h\or i\or j\or
      k\or l\or m\or n\or o\or p\or q\or r\or s\or t\or u\or v\or w\or
      x\or y\or z\else \@ctrerr\fi}}%
  \let\@Alph\@Alph@eng
  \let\@alph\@alph@eng
```

Unfortunately, the commands `\AA` and `\aa` are not encoding dependent in LaTeX (unlike e.g., `\oe` or `\DH`). They are defined as `\r{A}` and `\r{a}`. This leads to unpredictable results when the font encoding does not contain the Latin letters 'A' and 'a' (like `X2`).

```
  \DeclareTextSymbolDefault{\AA}{OT1}
  \DeclareTextSymbolDefault{\aa}{OT1}
  \DeclareTextCommand{\AA}{OT1}{\r A}
  \DeclareTextCommand{\aa}{OT1}{\r a}
\fi
```

The following block redefines the character class of uppercase Greek letters and some accents, if it is equal to 7 (variable family), to avoid incorrect results if the font encoding in some math family does not contain these characters in places of OT1 encoding. The code was taken from `amsmath.dtx`. See comments and further explanation there.

```
\begingroup\catcode`\"=12
% uppercase greek letters:
\def\@tempa#1{\expandafter\@tempb\meaning#1\relax\relax\relax\relax
 "0000\@nil#1}
\def\@tempb#1"#2#3#4#5#6\@nil#7{%
\ifnum"#2=7 \count@"1#3#4#5\relax
\ifnum\count@<"1000 \else \global\mathchardef#7="0#3#4#5\relax \fi
 \fi}
\@tempa\Gamma\@tempa\Delta\@tempa\Theta\@tempa\Lambda\@tempa\Xi
\@tempa\Pi\@tempa\Sigma\@tempa\Upsilon\@tempa\Phi\@tempa\Psi
\@tempa\Omega
% some accents:
\def\@tempa#1#2\@nil{\def\@tempc{#1}}\def\@tempb{\mathaccent}
\expandafter\@tempa\hat\relax\relax\@nil
\ifx\@tempb\@tempc
\def\@tempa#1\@nil{#1}%
\def\@tempb#1{\afterassignment\@tempa\mathchardef\@tempc=}%
\def\do#1"#2{}
\def\@tempd#1{\expandafter\@tempb#1\@nil
 \ifnum\@tempc>"FFF
 \xdef#1{\mathaccent"\expandafter\do\meaning\@tempc\space}%
 \fi}
\@tempd\hat\@tempd\check\@tempd\tilde\@tempd\acute\@tempd\grave
\@tempd\dot\@tempd\ddot\@tempd\breve\@tempd\bar
\fi
\endgroup
```

The user should use the `inputenc` package when any 8-bit Cyrillic font encoding is used, selecting one of the Cyrillic input encodings. We do not assume any default input encoding, so the user should explicitly call the `inputenc` package by `\usepackage{inputenc}`. We also removed `\AtBeginDocument`, so `inputenc` should be used before babel.

```
\@ifpackageloaded{inputenc}{}{%
\def\reserved@a{LWN}%
\ifx\reserved@a\cyrillicencoding\else
\def\reserved@a{OT2}%
\ifx\reserved@a\cyrillicencoding\else
\PackageWarning{babel}%
{No input encoding specified for Bulgarian language}\fi\fi}
```

Now we define two commands that offer the possibility to switch between Cyrillic and Roman encodings.

`\cyrillictext` `\latintext` The command `\cyrillictext` will switch from Latin font encoding to the Cyrillic font encoding, the command `\latintext` switches back. This assumes that the 'normal' font encoding is a Latin one. These commands are *declarations*, for shorter peaces of text the commands `\textlatin` and `\textcyrillic` can be used.

We comment out `\latintext` since it is defined in the core of babel (babel.def). We add the shorthand `\lat` for `\latintext`. Note that `\cyrillictext` has been defined above.

```
% \DeclareRobustCommand{\latintext}{%
% \fontencoding{\latinencoding}\selectfont
%   \def\encodingdefault{\latinencoding}}
\let\lat\latintext
```

`\textcyrillic` `\textlatin` These commands take an argument which is then typeset using the requested font encoding. `\textlatin` is commented out since it is defined in the core of babel. (It is defined there with `\DeclareRobustCommand` instead.)

```
\DeclareTextFontCommand{\textcyrillic}{\cyrillictext}
% \DeclareTextFontCommand{\textlatin}{\latintext}
```

The next step consists of defining commands to switch to (and from) the Bulgarian language.

`\captionsbulgarian` The macro `\captionsbulgarian` defines all strings used in the four standard document classes provided with LaTeX. The two commands `\cyr` and `\lat` activate Cyrillic resp. Latin encoding.

```
\addto\captionsbulgarian{%
  \def\prefacename{%
    {\cyr\CYRP\cyrr\cyre\cyrd\cyrg\cyro\cyrv\cyro\cyrr}}%
  \def\refname{%
    {\cyr\CYRL\cyri\cyrt\cyre\cyrr\cyra\cyrt\cyru\cyrr\cyra}}%
  \def\abstractname{%
    {\cyr\CYRA\cyrb\cyrs\cyrt\cyrr\cyra\cyrk\cyrt}}%
  \def\bibname{%
    {\cyr\CYRB\cyri\cyrb\cyrl\cyri\cyro\cyrg\cyrr\cyra\cyrf\cyri\cyrya}}%
  \def\chaptername{%
    {\cyr\CYRG\cyrl\cyra\cyrv\cyra}}%
  \def\appendixname{%
    {\cyr\CYRP\cyrr\cyri\cyrl\cyro\cyrzh\cyre\cyrn\cyri\cyre}}%
  \def\contentsname{%
    {\cyr\CYRS\cyrhrdsn\cyrd\cyrhrdsn\cyrr\cyrzh\cyra\cyrn\cyri\cyre}}%
  \def\listfigurename{%
    {\cyr\CYRS\cyrp\cyri\cyrs\cyrhrdsn\cyrk\ \cyrn\cyra\ \cyrf\cyri\cyrg\cyru\cyrr\cyri\cyrt
  \def\listtablename{%
    {\cyr\CYRS\cyrp\cyri\cyrs\cyrhrdsn\cyrk\ \cyrn\cyra\ \cyrt\cyra\cyrb\cyrl\cyri\cyrc\cyri
  \def\indexname{%
```

```
    {\cyr\CYRA\cyrz\cyrb\cyru\cyrch\cyre\cyrn\ \cyru\cyrk\cyra\cyrz\cyra\cyrt\cyre\cyrl}}%
  \def\authorname{%
    {\cyr\CYRI\cyrm\cyre\cyrn\cyre\cyrn\ \cyru\cyrk\cyra\cyrz\cyra\cyrt\cyre\cyrl}}%
  \def\figurename{%
    {\cyr\CYRF\cyri\cyrg\cyru\cyrr\cyra}}%
  \def\tablename{%
    {\cyr\CYRT\cyra\cyrb\cyrl\cyri\cyrc\cyra}}%
  \def\partname{%
    {\cyr\CYRCH\cyra\cyrs\cyrt}}%
  \def\enclname{%
    {\cyr\CYRP\cyrr\cyri\cyrl\cyro\cyrzh\cyre\cyrn\cyri\cyrya}}%
  \def\ccname{%
    {\cyr\cyrk\cyro\cyrp\cyri\cyrya}}%
  \def\headtoname{%
    {\cyr\CYRZ\cyra}}%
  \def\pagename{%
    {\cyr\CYRS\cyrt\cyrr.}}%
  \def\seename{%
    {\cyr\cyrv\cyrzh.}}%
  \def\alsoname{%
    {\cyr\cyrv\cyrzh.\ \cyrs\cyrhrdsn\cyrshch\cyro\ \cyri}}%
  \def\proofname{Proof}% <-- Needs translation
  \def\glossaryname{Glossary}% <-- Needs translation
}
```

`\datebulgarian` The macro `\datebulgarian` redefines the command `\today` to produce Bulgarian dates. It also provides the command `\todayRoman` which produces the date with the month in capital roman numerals, a popular format for dates in Bulgarian.

```
\def\datebulgarian{%
  \def\month@bulgarian{\ifcase\month\or
    \cyrya\cyrn\cyru\cyra\cyrr\cyri\or
    \cyrf\cyre\cyrv\cyrr\cyru\cyra\cyrr\cyri\or
    \cyrm\cyra\cyrr\cyrt\or
    \cyra\cyrp\cyrr\cyri\cyrl\or
    \cyrm\cyra\cyrishrt\or
    \cyryu\cyrn\cyri\or
    \cyryu\cyrl\cyri\or
    \cyra\cyrv\cyrg\cyru\cyrs\cyrt\or
    \cyrs\cyre\cyrp\cyrt\cyre\cyrm\cyrv\cyrr\cyri\or
    \cyro\cyrk\cyrt\cyro\cyrm\cyrv\cyrr\cyri\or
    \cyrn\cyro\cyre\cyrm\cyrv\cyrr\cyri\or
    \cyrd\cyre\cyrk\cyre\cyrm\cyrv\cyrr\cyri\fi}%
  \def\month@Roman{\expandafter\@Roman\month}%
  \def\today{\number\day~\month@bulgarian\ \number\year~\cyrg.}%
  \def\todayRoman{\number\day.\,\month@Roman.\,\number\year~\cyrg.}%
}
```

`\todayRoman` The month is often written with roman numbers in Bulgarian dates. Here we define date in this format:

```
\def\Romannumeral#1{\uppercase\expandafter{\romannumeral #1}}
\def\todayRoman{\number\day.\Romannumeral{\month}.\number\year~\cyrg.}
```

`\extrasbulgarian` The macro `\extrasbulgarian` will perform all the extra definitions needed for the Bulgarian language. The macro `\noextrasbulgarian` is used to cancel the actions of `\extrasbulgarian`.

The first action we define is to switch on the selected Cyrillic encoding whenever we enter 'bulgarian'.

```
\addto\extrasbulgarian{\cyrillictext}
```

When the encoding definition file was processed by LaTeX the current font encoding is stored in `\latinencoding`, assuming that LaTeX uses `T1` or `OT1` as default. Therefore we switch back to `\latinencoding` whenever the Bulgarian language is no longer 'active'.

```
\addto\noextrasbulgarian{\latintext}
```

For Bulgarian the `"` character also is made active.

```
\initiate@active@char{"}
```

The code above is necessary because we need extra active characters. The character `"` is used as indicated in table 29. We specify that the Bulgarian group of shorthands should be used.

```
\addto\extrasbulgarian{\languageshorthands{bulgarian}}
```

These characters are 'turned on' once, later their definition may vary.

```
\addto\extrasbulgarian{%
  \bbl@activate{"}}
\addto\noextrasbulgarian{%
  \bbl@deactivate{"}}
```

The `X2` and `T2*` encodings do not contain `spanish_shriek` and `spanish_query` symbols; as a consequence, the ligatures '`?‘`' and '`!‘`' do not work with them (these characters are useless for Cyrillic texts anyway). But we define the shorthands to emulate these ligatures (optionally).

We do not use `\latinencoding` here (but instead explicitly use `OT1`) because the user may choose `T2A` to be the primary encoding, but it does not contain these characters.

```
⟨*spanishligs⟩
\declare@shorthand{bulgarian}{?‘}{\UseTextSymbol{OT1}\textquestiondown}
\declare@shorthand{bulgarian}{!‘}{\UseTextSymbol{OT1}\textexclamdown}
⟨/spanishligs⟩
```

To be able to define the function of '`"`', we first define a couple of 'support' macros.

`\dq` We save the original double quote character in `\dq` to keep it available, the math accent `\"`can now be typed as '`"`'.

```
\begingroup \catcode‘\"12
```

```
\def\reserved@a{\endgroup
  \def\@SS{\mathchar"7019}
  \def\dq{"}}
\reserved@a
```

Now we can define the doublequote macros: german and french quotes. We use definitions of these quotes made in babel.sty. The french quotes are contained in the `T2*` encodings.

```
\declare@shorthand{bulgarian}{"`}{\glqq}
\declare@shorthand{bulgarian}{"'}{\grqq}
\declare@shorthand{bulgarian}{"<}{\flqq}
\declare@shorthand{bulgarian}{">}{\frqq}
```

Some additional commands:

```
\declare@shorthand{bulgarian}{""}{\hskip\z@skip}
\declare@shorthand{bulgarian}{"~}{\textormath{\leavevmode\hbox{-}}{-}}
\declare@shorthand{bulgarian}{"=}{\nobreak-\hskip\z@skip}
\declare@shorthand{bulgarian}{"|}{%
\textormath{\nobreak\discretionary{-}{}{\kern.03em}%
\allowhyphens}{}}
```

The next two macros for `"-` and `"---` are somewhat different. We must check whether the second token is a hyphen character:

```
\declare@shorthand{bulgarian}{"-}{%
```

If the next token is '`-`', we typeset an emdash, otherwise a hyphen sign:

```
  \def\bulgarian@sh@tmp{%
    \if\bulgarian@sh@next-\expandafter\bulgarian@sh@emdash
    \else\expandafter\bulgarian@sh@hyphen\fi
  }%
```

TEX looks for the next token after the first '`-`': the meaning of this token is written to `\bulgarian@sh@next` and `\bulgarian@sh@tmp` is called.

```
  \futurelet\bulgarian@sh@next\bulgarian@sh@tmp}
```

Here are the definitions of hyphen and emdash. First the hyphen:

```
\def\bulgarian@sh@hyphen{\nobreak\-\bbl@allowhyphens}
```

For the emdash definition, there are the two parameters: we must 'eat' two last hyphen signs of our emdash . . . :

```
\def\bulgarian@sh@emdash#1#2{\cdash-#1#2}
```

`\cdash` . . . these two parameters are useful for another macro: `\cdash`:

```
\ifx\cdash\undefined % should be defined earlier
\def\cdash#1#2#3{\def\tempx@{#3}%
\def\tempa@{-}\def      empb@{~}\def    empc@{*}%
 \ifx\tempx@\tempa@\@Acdash\else
  \ifx\tempx@\tempb@\@Bcdash\else
   \ifx\tempx@\tempc@\@Ccdash\else
    \errmessage{Wrong usage of cdash}\fi\fi\fi}
```

second parameter (or third for `\cdash`) shows what kind of emdash to create in next step

`"---` ordinary (plain) Cyrillic emdash inside text: an unbreakable thinspace will be inserted before only in case of a *space* before the dash (it is necessary for dashes after display maths formulae: there could be lists, enumerations etc. started with "—where $a$ is ..." i.e., the dash starts a line). (Firstly there were planned rather soft rules for user:he may put a space before the dash or not. But it is difficult to place this thinspace automatically, i.e., by checking modes because after display formulae TEX uses horizontal mode. Maybe there is a misunderstanding? Maybe there is another way?) After a dash a breakable thinspace is always placed;

```
% What is more grammatically: .2em or .2\fontdimen6\font?
\def\@Acdash{\ifdim\lastskip>\z@\unskip\nobreak\hskip.2em\fi
\cyrdash\hskip.2em\ignorespaces}%
```

`"--~` emdash in compound names or surnames (like Mendeleev–Klapeiron); this dash has no space characters around; after the dash some space is added `\exhyphenalty`

```
\def\@Bcdash{\leavevmode\ifdim\lastskip>\z@\unskip\fi
 \nobreak\cyrdash\penalty\exhyphenpenalty\hskip\z@skip\ignorespaces}%
```

`"--*` for denoting direct speech (a space like `\enskip` must follow the emdash);

```
\def\@Ccdash{\leavevmode
 \nobreak\cyrdash\nobreak\hskip.35em\ignorespaces}%
%\fi
```

`\cyrdash` Finally the macro for "body" of the Cyrillic emdash. The `\cyrdash` macro will be defined in case this macro hasn't been defined in a fontenc file. For T2*fonts, cyrdash will be placed in the code of the English emdash thus it uses ligature `---`.

```
% Is there an IF necessary?
\ifx\cyrdash\undefined
\def\cyrdash{\hbox to.8em{--\hss--}}
\fi
```

Here a really new macro—to place thinspace between initials. This macro used instead of `\,` allows hyphenation in the following surname.

```
\declare@shorthand{bulgarian}{",}{\nobreak\hskip.2em\ignorespaces}
```

The Bulgarian hyphenation patterns can be used with `\lefthyphenmin` and `\righthyphenmin` set to 2.

```
\providehyphenmins{\CurrentOption}{\tw@\tw@}
\fi
```

Now the action `\extrasbulgarian` has to execute is to make sure that the command `\frenchspacing` is in effect. If this is not the case the execution of `\noextrasbulgarian` will switch it off again.

```
\addto\extrasbulgarian{\bbl@frenchspacing}
\addto\noextrasbulgarian{\bbl@nonfrenchspacing}
```

Make the double quotes produce the traditional quotes used in Bulgarian texts (these are the German quotes).

```
% \initiate@active@char{`}
%  \initiate@active@char{'}
% \addto\extrasbulgarian{%
%   \bbl@activate{`}}
% \addto\extrasbulgarian{%
%   \bbl@activate{'}}
% \addto\noextrasbulgarian{%
%   \bbl@deactivate{`}}
% \addto\noextrasbulgarian{%
%   \bbl@deactivate{'}}
% \def\mlron{\bbl@activate{`}\bbl@activate{'}}
% \def\mlroff{\bbl@deactivate{`}\bbl@deactivate{'}}
% \declare@shorthand{bulgarian}{``}{\glqq}
% \declare@shorthand{bulgarian}{''}{\grqq}
```

Next we add a new enumeration style for Bulgarian manuscripts with Cyrillic letters,and later on we define some math operator names in accordance with Bulgarian typesetting traditions.

`\@Alph@bul` We begin by defining `\@Alph@bul` which works like `\@Alph`, but produces (uppercase) Cyrillic letters intead of Latin ones. The letters ISHRT, HRDSN and SFTSN are skipped, as usual for such enumeration.

```
\def\enumBul{\let\@Alph\@Alph@bul \let\@alph\@alph@bul}
\def\enumEng{\let\@Alph\@Alph@eng \let\@alph\@alph@eng}
\def\enumLat{\let\@Alph\@Alph@eng \let\@alph\@alph@eng}
\addto\extrasbulgarian{\enumBul}
\addto\noextrasbulgarian{\enumLat}
\def\@Alph@bul#1{%
  \ifcase#1\or
  \CYRA\or \CYRB\or \CYRV\or \CYRG\or \CYRD\or \CYRE\or \CYRZH\or
  \CYRZ\or \CYRI\or \CYRK\or \CYRL\or \CYRM\or \CYRN\or \CYRO\or
  \CYRP\or \CYRR\or \CYRS\or \CYRT\or \CYRU\or \CYRF\or \CYRH\or
  \CYRC\or \CYRCH\or \CYRSH\or \CYRSHCH\or \CYRYU\or \CYRYA\else
  \@ctrerr\fi
 }
\def\@Alph@eng#1{%
  \ifcase#1\or
  A\or B\or C\or D\or E\or F\or G\or H\or I\or J\or K\or L\or M\or
  N\or O\or P\or Q\or R\or S\or T\or U\or V\or W\or X\or Y\or Z\else
  \@ctrerr\fi
  }
```

`\@alph@bul` The macro `\@alph@bul` is similar to `\@Alph@bul`; it produces lowercase Bulgarian letters.

```
\def\@alph@bul#1{%
  \ifcase#1\or
  \cyra\or \cyrb\or \cyrv\or \cyrg\or \cyrd\or \cyre\or \cyrzh\or
  \cyrz\or \cyri\or \cyrk\or \cyrl\or \cyrm\or \cyrn\or \cyro\or
  \cyrp\or \cyrr\or \cyrs\or \cyrt\or \cyru\or \cyrf\or \cyrh\or
  \cyrc\or \cyrch\or \cyrsh\or \cyrshch\or \cyryu\or \cyrya\else
  \@ctrerr\fi
  }
\def\@alph@eng#1{%
  \ifcase#1\or
  a\or b\or c\or d\or e\or f\or g\or h\or i\or j\or k\or l\or m\or
  n\or o\or p\or q\or r\or s\or t\or u\or v\or w\or x\or y\or z\else
  \@ctrerr\fi
  }
```

Set up default Cyrillic math alphabets. To use Cyrillic letters in math mode user should load the `textmath` package *before* loading fontenc package (or babel). Note,that by default Cyrillic letters are taken from upright font in math mode (unlike Latin letters).

```
%\RequirePackage{textmath}
\@ifundefined{sym\cyrillicencoding letters}{}{%
\SetSymbolFont{\cyrillicencoding letters}{bold}\cyrillicencoding
  \rmdefault\bfdefault\updefault
\DeclareSymbolFontAlphabet\cyrmathrm{\cyrillicencoding letters}
```

And we need a few commands to be able to switch to different variants.

```
\DeclareMathAlphabet\cyrmathbf\cyrillicencoding
  \rmdefault\bfdefault\updefault
\DeclareMathAlphabet\cyrmathsf\cyrillicencoding
  \sfdefault\mddefault\updefault
\DeclareMathAlphabet\cyrmathit\cyrillicencoding
  \rmdefault\mddefault\itdefault
\DeclareMathAlphabet\cyrmathtt\cyrillicencoding
  \ttdefault\mddefault\updefault
\SetMathAlphabet\cyrmathsf{bold}\cyrillicencoding
  \sfdefault\bfdefault\updefault
\SetMathAlphabet\cyrmathit{bold}\cyrillicencoding
  \rmdefault\bfdefault\itdefault
}
```

Some math functions in Bulgarian math books have other names: e.g., `sinh` in Bulgarian is written as `sh` etc. So we define a number of new math operators.

`\sinh`:

```
\def\sh{\mathop{\operator@font sh}\nolimits}
```

`\cosh`:

```
\def\ch{\mathop{\operator@font ch}\nolimits}
```

`\tan`:

```
\def\tg{\mathop{\operator@font tg}\nolimits}
```

`\arctan`:

```
\def\arctg{\mathop{\operator@font arctg}\nolimits}
```

`\arccot`:

```
\def\arcctg{\mathop{\operator@font arcctg}\nolimits}
```

The following macro conflicts with `\th` defined in Latin 1 encoding: `\tanh`:

```
\addto\extrasrussian{%
  \babel@save{\th}%
  \let\ltx@th\th
  \def\th{\textormath{\ltx@th}%
                      {\mathop{\operator@font th}\nolimits}}%
  }
```

`\cot`:

```
\def\ctg{\mathop{\operator@font ctg}\nolimits}
```

`\coth`:

```
\def\cth{\mathop{\operator@font cth}\nolimits}
```

`\csc`:

```
\def\cosec{\mathop{\operator@font cosec}\nolimits}
```

This is for compatibility with older Bulgarian packages.

```
\DeclareRobustCommand{\No}{%
    \ifmmode{\nfss@text{\textnumero}}\else\textnumero\fi}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{bulgarian}
⟨/code⟩
```

## 56 The Ukrainian language

The file `ukraineb.dtx`[64] defines all the language-specific macros for the Ukrainian language. It needs the file `cyrcod` for success documentation with Ukrainian encodings (see below).

For this language the character `"` is made active. In table 31 an overview is given of its purpose.

| | |
|---|---|
| `"\|` | disable ligature at this position. |
| `"-` | an explicit hyphen sign, allowing hyphenation in the rest of the word. |
| `"---` | Cyrillic emdash in plain text. |
| `"--~` | Cyrillic emdash in compound names (surnames). |
| `"--*` | Cyrillic emdash for denoting direct speech. |
| `""` | like `"-`, but producing no hyphen sign (for compund words with hyphen, e.g. `x-""y` or some other signs as “disable/enable”). |
| `"~` | for a compound word mark without a breakpoint. |
| `"=` | for a compound word mark with a breakpoint, allowing hyphenation in the composing words. |
| `",` | thinspace for initials with a breakpoint in following surname. |
| ``"` `` | for German left double quotes (looks like „). |
| `"'` | for German right double quotes (looks like “). |
| ” `"<` | for French left double quotes (looks like ≪). |
| `">` | for French right double quotes (looks like ≫). |

Table 31: The extra definitions made by `ukraineb`

The quotes in table 31 (see, also table 27) can also be typeset by using the commands in table 32 (see, also table 28).

| | |
|---|---|
| `\cdash---` | Cyrillic emdash in plain text. |
| `\cdash--~` | Cyrillic emdash in compound names (surnames). |
| `\cdash--*` | Cyrillic emdash for denoting direct speech. |
| `\glqq` | for German left double quotes (looks like „). |
| `\grqq` | for German right double quotes (looks like “). |
| ” `\flqq` | for French left double quotes (looks like ≪). |
| `\frqq` | for French right double quotes (looks like ≫). |
| `\dq` | the original quotes character (`"`). |

Table 32: More commands which produce quotes, defined by babel

[64]The file described in this section has version number ?. This file was derived from the `russianb.dtx` version 1.1g.

The French quotes are also available as ligatures '<<' and '>>' in 8-bit Cyrillic font encodings (LCY, X2, T2*) and as '<' and '>' characters in 7-bit Cyrillic font encodings (OT2 and LWN).

The quotation marks traditionally used in Ukrainian and Russian languages were borrowed from other languages (e.g. French and German) so they keep their original names.

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
⟨*code⟩
\LdfInit{ukrainian}{captionsukrainian}
```

When this file is read as an option, i.e., by the `\usepackage` command, `ukraineb` will be an 'unknown' language, in which case we have to make it known. So we check for the existence of `\l@ukrainian` to see whether we have to do something here.

```
\ifx\l@ukrainian\@undefined
  \@nopatterns{Ukrainian}
  \adddialect\l@ukrainian0
\fi
```

`\latinencoding` We need to know the encoding for text that is supposed to be which is active at the end of the babel package. If the `fontenc` package is loaded later, then... too bad!

```
\let\latinencoding\cf@encoding
```

The user may choose between different available Cyrillic encodings—e.g., X2, LCY, or LWN. Hopefully, X2 will eventually replace the two latter encodings (LCY and LWN). If the user wants to use another font encoding than the default (T2A), he has to load the corresponding file *before* `ukraineb.sty`. This may be done in the following way:

```
% override the default X2 encoding used in Babel
\usepackage[LCY,OT1]{fontenc}
\usepackage[english,ukrainian]{babel}
```

Note: for the Ukrainian language, the T2A encoding is better than X2, because X2 does not contain Latin letters, and users should be very careful to switch the language every time they want to typeset a Latin word inside a Ukrainian phrase or vice versa.

We parse the `\cdp@list` containing the encodings known to LaTeX in the order they were loaded. We set the `\cyrillicencoding` to the *last* loaded encoding in the list of supported Cyrillic encodings: OT2, LWN, LCY, X2, T2C, T2B, T2A, if any.

```
\def\reserved@a#1#2{%
  \edef\reserved@b{#1}%
  \edef\reserved@c{#2}%
  \ifx\reserved@b\reserved@c
    \let\cyrillicencoding\reserved@c
```

```
   \fi}
\def\cdp@elt#1#2#3#4{%
   \reserved@a{#1}{OT2}%
   \reserved@a{#1}{LWN}%
   \reserved@a{#1}{LCY}%
   \reserved@a{#1}{X2}%
   \reserved@a{#1}{T2C}%
   \reserved@a{#1}{T2B}%
   \reserved@a{#1}{T2A}}
\cdp@list
```

Now, if `\cyrillicencoding` is undefined, then the user did not load any of supported encodings. So, we have to set `\cyrillicencoding` to some default value. We test the presence of the encoding definition files in the order from less preferable to more preferable encodings. We use the lowercase names (i.e., `lcyenc.def` instead of `LCYenc.def`).

```
\ifx\cyrillicencoding\undefined
  \IfFileExists{ot2enc.def}{\def\cyrillicencoding{OT2}}\relax
  \IfFileExists{lwnenc.def}{\def\cyrillicencoding{LWN}}\relax
  \IfFileExists{lcyenc.def}{\def\cyrillicencoding{LCY}}\relax
  \IfFileExists{x2enc.def}{\def\cyrillicencoding{X2}}\relax
  \IfFileExists{t2cenc.def}{\def\cyrillicencoding{T2C}}\relax
  \IfFileExists{t2benc.def}{\def\cyrillicencoding{T2B}}\relax
  \IfFileExists{t2aenc.def}{\def\cyrillicencoding{T2A}}\relax
```

If `\cyrillicencoding` is still undefined, then the user seems not to have a properly installed distribution. A fatal error.

```
  \ifx\cyrillicencoding\undefined
    \PackageError{babel}%
      {No Cyrillic encoding definition files were found}%
      {Your installation is incomplete.\MessageBreak
       You need at least one of the following files:\MessageBreak
       \space\space
       x2enc.def, t2aenc.def, t2benc.def, t2cenc.def,\MessageBreak
       \space\space
       lcyenc.def, lwnenc.def, ot2enc.def.}%
  \else
```

We avoid `\usepackage[\cyrillicencoding]{fontenc}` because we don't want to force the switch of `\encodingdefault`.

```
    \lowercase
      \expandafter{\expandafter\input\cyrillicencoding enc.def\relax}%
  \fi
\fi
```

```
    \PackageInfo{babel}
      {Using `\cyrillicencoding' as a default Cyrillic encoding}%
```

```
\DeclareRobustCommand{\Ukrainian}{%
  \fontencoding\cyrillicencoding\selectfont
  \let\encodingdefault\cyrillicencoding
  \expandafter\set@hyphenmins\ukrainianhyphenmins
  \language\l@ukrainian}%
\DeclareRobustCommand{\English}{%
  \fontencoding\latinencoding\selectfont
  \let\encodingdefault\latinencoding
  \expandafter\set@hyphenmins\englishhyphenmins
  \language\l@english}%
\let\Ukr\Ukrainian
\let\Eng\English
\let\cyrillictext\Ukrainian
\let\cyr\Ukrainian
```

Since the `X2` encoding does not contain Latin letters, we should make some redefinitions of LaTeX macros which implicitly produce Latin letters.

```
\expandafter\ifx\csname T@X2\endcsname\relax\else
```

We put `\latinencoding` in braces to avoid problems with `\@alph` inside minipages (e.g., footnotes inside minipages) where `\@alph` is expanded and we get for example '`\fontencoding OT1`' (`\fontencoding` is robust).

```
  \def\@alph#1{{\fontencoding{\latinencoding}\selectfont
    \ifcase#1\or
      a\or b\or c\or d\or e\or f\or g\or h\or
      i\or j\or k\or l\or m\or n\or o\or p\or
      q\or r\or s\or t\or u\or v\or w\or x\or
      y\or z\else\@ctrerr\fi}}%
  \def\@Alph#1{{\fontencoding{\latinencoding}\selectfont
    \ifcase#1\or
      A\or B\or C\or D\or E\or F\or G\or H\or
      I\or J\or K\or L\or M\or N\or O\or P\or
      Q\or R\or S\or T\or U\or V\or W\or X\or
      Y\or Z\else\@ctrerr\fi}}%
```

Unfortunately, the commands `\AA` and `\aa` are not encoding dependent in LaTeX (unlike e.g., `\oe` or `\DH`). They are defined as `\r{A}` and `\r{a}`. This leads to unpredictable results when the font encoding does not contain the Latin letters 'A' and 'a' (like `X2`).

```
  \DeclareTextSymbolDefault{\AA}{OT1}
  \DeclareTextSymbolDefault{\aa}{OT1}
  \DeclareTextCommand{\aa}{OT1}{\r a}
  \DeclareTextCommand{\AA}{OT1}{\r A}
\fi
```

The following block redefines the character class of uppercase Greek letters and some accents, if it is equal to 7 (variable family), to avoid incorrect results if the font encoding in some math family does not contain these characters in places of OT1 encoding. The code was taken from `amsmath.dtx`. See comments and further explanation there.

```
% \begingroup\catcode`\"=12
% % uppercase greek letters:
% \def\@tempa#1{\expandafter\@tempb\meaning#1\relax\relax\relax\relax
%   "0000\@nil#1}
% \def\@tempb#1"#2#3#4#5#6\@nil#7{%
%   \ifnum"#2=7 \count@"1#3#4#5\relax
%     \ifnum\count@<"1000 \else \global\mathchardef#7="0#3#4#5\relax \fi
%   \fi}
% \@tempa\Gamma\@tempa\Delta\@tempa\Theta\@tempa\Lambda\@tempa\Xi
% \@tempa\Pi\@tempa\Sigma\@tempa\Upsilon\@tempa\Phi\@tempa\Psi
% \@tempa\Omega
% % some accents:
% \def\@tempa#1#2\@nil{\def\@tempc{#1}}\def\@tempb{\mathaccent}
% \expandafter\@tempa\hat\relax\relax\@nil
% \ifx\@tempb\@tempc
%   \def\@tempa#1\@nil{#1}%
%   \def\@tempb#1{\afterassignment\@tempa\mathchardef\@tempc=}%
%   \def\do#1"#2{}
%   \def\@tempd#1{\expandafter\@tempb#1\@nil
%     \ifnum\@tempc>"FFF
%       \xdef#1{\mathaccent"\expandafter\do\meaning\@tempc\space}%
%     \fi}
%   \@tempd\hat\@tempd\check\@tempd\tilde\@tempd\acute\@tempd\grave
%   \@tempd\dot\@tempd\ddot\@tempd\breve\@tempd\bar
% \fi
% \endgroup
```

The user must use the `inputenc` package when any 8-bit Cyrillic font encoding is used, selecting one of the Cyrillic input encodings. We do not assume any default input encoding, so the user should explicitly call the `inputenc` package by `\usepackage{inputenc}`. We also removed `\AtBeginDocument`, so `inputenc` should be used before babel.

```
\@ifpackageloaded{inputenc}{}{%
  \def\reserved@a{LWN}%
  \ifx\reserved@a\cyrillicencoding\else
    \def\reserved@a{OT2}%
    \ifx\reserved@a\cyrillicencoding\else
      \PackageWarning{babel}%
        {No input encoding specified for Ukrainian language}
  \fi\fi}
```

Now we define two commands that offer the possibility to switch between Cyrillic and Roman encodings.

`\cyrillictext` `\latintext` The command `\cyrillictext` will switch from Latin font encoding to the Cyrillic font encoding, the command `\latintext` switches back. This assumes that the 'normal' font encoding is a Latin one. These commands are *declarations*, for shorter peaces of text the commands `\textlatin` and `\textcyrillic` can be used.

```
%\DeclareRobustCommand{\latintext}{%
%  \fontencoding{\latinencoding}\selectfont
%  \def\encodingdefault{\latinencoding}}
\let\lat\latintext
```

`\textcyrillic` `\textlatin` These commands take an argument which is then typeset using the requested font encoding.

```
\DeclareTextFontCommand{\textcyrillic}{\cyrillictext}
%\DeclareTextFontCommand{\textlatin}{\latintext}
```

We make the TEX

```
%\ifx\ltxTeX\undefined\let\ltxTeX\TeX\fi
%\ProvideTextCommandDefault{\TeX}{\textlatin{\ltxTeX}}
```

and LATEX logos encoding independent.

```
%\ifx\ltxLaTeX\undefined\let\ltxLaTeX\LaTeX\fi
%\ProvideTextCommandDefault{\LaTeX}{\textlatin{\ltxLaTeX}}
```

The next step consists of defining commands to switch to (and from) the Ukrainian language.

`\captionsukrainian` The macro `\captionsukrainian` defines all strings used in the four standard document classes provided with LATEX. The two commands `\cyr` and `\lat` activate Cyrillic resp. Latin encoding.

```
\addto\captionsukrainian{%
  \def\prefacename{{\cyr\CYRV\cyrs\cyrt\cyru\cyrp}}%
% \def\prefacename{{\cyr\CYRP\cyre\cyrr\cyre\cyrd\cyrm\cyro\cyrv\cyra}}%
  \def\refname{%
    {\cyr\CYRL\cyrii\cyrt\cyre\cyrr\cyra\cyrt\cyru\cyrr\cyra}}%
%  \def\refname{%
%    {\cyr\CYRP\cyre\cyrr\cyre\cyrl\cyrii\cyrk
%         \ \cyrp\cyro\cyrs\cyri\cyrl\cyra\cyrn\cyrsftsn}}%
  \def\abstractname{%
    {\cyr\CYRA\cyrn\cyro\cyrt\cyra\cyrc\cyrii\cyrya}}%
%  \def\abstractname{{\cyr\CYRR\cyre\cyrf\cyre\cyrr\cyra\cyrt}}%
  \def\bibname{%
    {\cyr\CYRB\cyrii\cyrb\cyrl\cyrii\cyro\cyrgup\cyrr\cyra\cyrf\cyrii\cyrya}}%
% \def\bibname{{\cyr\CYRL\cyrii\cyrt\cyre\cyrr\cyra\cyrt\cyru\cyrr\cyra}}%
  \def\chaptername{{\cyr\CYRR\cyro\cyrz\cyrd\cyrii\cyrl}}%
%  \def\chaptername{{\cyr\CYRG\cyrl\cyra\cyrv\cyra}}%
  \def\appendixname{{\cyr\CYRD\cyro\cyrd\cyra\cyrt\cyro\cyrk}}%
  \def\contentsname{{\cyr\CYRZ\cyrm\cyrii\cyrs\cyrt}}%
  \def\listfigurename{{\cyr\CYRP\cyre\cyrr\cyre\cyrl\cyrii\cyrk
         \ \cyrii\cyrl\cyryu\cyrs\cyrt\cyrr\cyra\cyrc\cyrii\cyrishrt}}%
  \def\listtablename{{\cyr\CYRP\cyre\cyrr\cyre\cyrl\cyrii\cyrk
         \ \cyrt\cyra\cyrb\cyrl\cyri\cyrc\cyrsftsn}}%
  \def\indexname{{\cyr\CYRP\cyro\cyrk\cyra\cyrzh\cyrch\cyri\cyrk}}%
  \def\authorname{{\cyr\CYRII\cyrm\cyre\cyrn\cyrn\cyri\cyrishrt
         \ \cyrp\cyro\cyrk\cyra\cyrzh\cyrch\cyri\cyrk}}%
  \def\figurename{{\cyr\CYRR\cyri\cyrs.}}%
%  \def\figurename{\cyr\CYRR\cyri\cyrs\cyru\cyrn\cyro\cyrk}}%
   \def\tablename{{\cyr\CYRT\cyra\cyrb\cyrl.}}%
%  \def\tablename{\cyr\CYRT\cyra\cyrb\cyrl\cyri\cyrc\cyrya}}%
   \def\partname{{\cyr\CYRCH\cyra\cyrs\cyrt\cyri\cyrn\cyra}}%
   \def\enclname{{\cyr\cyrv\cyrk\cyrl\cyra\cyrd\cyrk\cyra}}%
   \def\ccname{{\cyr\cyrk\cyro\cyrp\cyrii\cyrya}}%
   \def\headtoname{{\cyr\CYRD\cyro}}%
   \def\pagename{{\cyr\cyrs.}}%
%  \def\pagename{{\cyr\cyrs\cyrt\cyro\cyrr\cyrii\cyrn\cyrk\cyra}}%
   \def\seename{{\cyr\cyrd\cyri\cyrv.}}%
   \def\alsoname{{\cyr\cyrd\cyri\cyrv.\ \cyrt\cyra\cyrk\cyro\cyrzh}}
   \def\proofname{{\cyr\CYRD\cyro\cyrv\cyre\cyrd\cyre\cyrn\cyrn\cyrya}}%
   \def\glossaryname{{\cyr\CYRS\cyrl\cyro\cyrv\cyrn\cyri\cyrk\ %
                    \cyrt\cyre\cyrr\cyrm\cyrii\cyrn\cyrii\cyrv}}%
   }
```

`\dateukrainian` The macro `\dateukrainian` redefines the command `\today` to produce Ukrainian dates.

```
\def\dateukrainian{%
   \def\today{\number\day~\ifcase\month\or
     \cyrs\cyrii\cyrch\cyrn\cyrya\or
     \cyrl\cyryu\cyrt\cyro\cyrg\cyro\or
     \cyrb\cyre\cyrr\cyre\cyrz\cyrn\cyrya\or
     \cyrk\cyrv\cyrii\cyrt\cyrn\cyrya\or
     \cyrt\cyrr\cyra\cyrv\cyrn\cyrya\or
     \cyrch\cyre\cyrr\cyrv\cyrn\cyrya\or
     \cyrl\cyri\cyrp\cyrn\cyrya\or
     \cyrs\cyre\cyrr\cyrp\cyrn\cyrya\or
     \cyrv\cyre\cyrr\cyre\cyrs\cyrn\cyrya\or
     \cyrzh\cyro\cyrv\cyrt\cyrn\cyrya\or
     \cyrl\cyri\cyrs\cyrt\cyro\cyrp\cyra\cyrd\cyra\or
     \cyrg\cyrr\cyru\cyrd\cyrn\cyrya\fi
     \space\number\year~\cyrr.}}
```

`\extrasukrainian` The macro `\extrasukrainian` will perform all the extra definitions needed for the Ukrainian language. The macro `\noextrasukrainian` is used to cancel the actions of `\extrasukrainian`.

The first action we define is to switch on the selected Cyrillic encoding whenever we enter 'ukrainian'.

```
\addto\extrasukrainian{\cyrillictext}
```

When the encoding definition file was processed by LaTeX the current font encoding is stored in `\latinencoding`, assuming that LaTeX uses `T1` or `OT1` as default. Therefore we switch back to `\latinencoding` whenever the Ukrainian language is no longer 'active'.

```
\addto\noextrasukrainian{\latintext}
```

Next we must allow hyphenation in the Ukrainian words with apostrophe whenever we enter 'ukrainian'. This solution was proposed by Vladimir Volovich <vvv@vvv.vsu.ru>

```
\addto\extrasukrainian{\lccode`\'=`\'}
\addto\noextrasukrainian{\lccode`\'=0}
```

`\verbatim@font` In order to get both Latin and Cyrillic letters in verbatim text we need to change the definition of an internal LaTeX command somewhat:

```
%\def\verbatim@font{%
%  \let\encodingdefault\latinencoding
%  \normalfont\ttfamily
%  \expandafter\def\csname\cyrillicencoding-cmd\endcsname##1##2{%
%    \ifx\protect\@typeset@protect
%      \begingroup\UseTextSymbol\cyrillicencoding##1\endgroup
%    \else\noexpand##1\fi}}
```

The category code of the characters '`:`', '`;`', '`!`', and '`?`' is made `\active` to insert a little white space.

For Ukrainian (as well as for Russian and German) the `"` character also is made active.

Note: It is *very* questionable whether the Russian typesetting tradition requires additional spacing before those punctuation signs. Therefore, we make the corresponding code optional. If you need it, then define the `frenchpunct` docstrip option in `babel.ins`.

Borrowed from french. Some users dislike automatic insertion of a space before 'double punctuation', and prefer to decide themselves whether a space should be added or not; so a hook `\NoAutoSpaceBeforeFDP` is provided: if this command is added (in file `ukraineb.cfg`, or anywhere in a document) `ukraineb` will respect your typing, and introduce a suitable space before 'double punctuation' *if and only if* a space is typed in the source file before those signs.

The command `\AutoSpaceBeforeFDP` switches back to the default behavior of `ukraineb`.

```
⟨*frenchpunct⟩
\initiate@active@char{:}
\initiate@active@char{;}
⟨/frenchpunct⟩
⟨*frenchpunct | spanishligs⟩
\initiate@active@char{!}
\initiate@active@char{?}
⟨/frenchpunct | spanishligs⟩
\initiate@active@char{"}
```

The code above is necessary because we need extra active characters. The character `"` is used as indicated in table 31.

We specify that the Ukrainian group of shorthands should be used.

```
\addto\extrasukrainian{\languageshorthands{ukrainian}}
```

These characters are 'turned on' once, later their definition may vary.

```
\addto\extrasukrainian{%
⟨frenchpunct⟩  \bbl@activate{:}\bbl@activate{;}%
⟨frenchpunct | spanishligs⟩  \bbl@activate{!}\bbl@activate{?}%
```

```
  \bbl@activate{"}}
\addto\noextrasukrainian{%
⟨frenchpunct⟩   \bbl@deactivate{:}\bbl@deactivate{;}%
⟨frenchpunct | spanishligs⟩   \bbl@deactivate{!}\bbl@deactivate{?}%
  \bbl@deactivate{"}}
```

The X2 and T2* encodings do not contain spanish_shriek and spanish_query symbols; as a consequence, the ligatures '?`' and '!`' do not work with them (these characters are useless for Cyrillic texts anyway). But we define the shorthands to emulate these ligatures (optionally).

We do not use \latinencoding here (but instead explicitly use OT1) because the user may choose T2A to be the primary encoding, but it does not contain these characters.

```
⟨*spanishligs⟩
\declare@shorthand{ukrainian}{?`}{\UseTextSymbol{OT1}\textquestiondown}
\declare@shorthand{ukrainian}{!`}{\UseTextSymbol{OT1}\textexclamdown}
⟨/spanishligs⟩
```

\ukrainian@sh@;@
\ukrainian@sh@:@
\ukrainian@sh@!@
\ukrainian@sh@?@

We have to reduce the amount of white space before ;, : and !. This should only happen in horizontal mode, hence the test with \ifhmode.

```
⟨*frenchpunct⟩
\declare@shorthand{ukrainian}{;}{%
  \ifhmode
```

In horizontal mode we check for the presence of a 'space', 'unskip' if it exists and place a 0.1em kerning.

```
    \ifdim\lastskip>\z@
      \unskip\nobreak\kern.1em
    \else
```

If no space has been typed, we add \FDP@thinspace which will be defined, up to the user's wishes, as an automatic added thinspace, or as \@empty.

```
        \FDP@thinspace
    \fi
  \fi
```

Now we can insert a ';' character.

```
  \string;}
```

The other definitions are very similar.

```
\declare@shorthand{ukrainian}{:}{%
  \ifhmode
    \ifdim\lastskip>\z@
      \unskip\nobreak\kern.1em
    \else
        \FDP@thinspace
    \fi
  \fi
  \string:}
```

```
\declare@shorthand{ukrainian}{!}{%
  \ifhmode
    \ifdim\lastskip>\z@
      \unskip\nobreak\kern.1em
    \else
        \FDP@thinspace
    \fi
  \fi
  \string!}
```

```
\declare@shorthand{ukrainian}{?}{%
  \ifhmode
    \ifdim\lastskip>\z@
      \unskip\nobreak\kern.1em
    \else
        \FDP@thinspace
    \fi
  \fi
  \string?}
```

\AutoSpaceBeforeFDP
\NoAutoSpaceBeforeFDP
\FDP@thinspace

\FDP@thinspace is defined as unbreakable spaces if \AutoSpaceBeforeFDP is activated or as \@empty if \NoAutoSpaceBeforeFDP is in use. The default is \AutoSpaceBeforeFDP.

```
\def\AutoSpaceBeforeFDP{%
      \def\FDP@thinspace{\nobreak\kern.1em}}
\def\NoAutoSpaceBeforeFDP{\let\FDP@thinspace\@empty}
\AutoSpaceBeforeFDP
```

\FDPon
\FDPoff

The next macros allow to switch on/off activeness of double punctuation signs.

```
\def\FDPon{\bbl@activate{:}%
        \bbl@activate{;}%
        \bbl@activate{?}%
        \bbl@activate{!}}
\def\FDPoff{\bbl@deactivate{:}%
        \bbl@deactivate{;}%
        \bbl@deactivate{?}%
        \bbl@deactivate{!}}
```

\system@sh@:@
\system@sh@!@
\system@sh@?@
\system@sh@;@

When the active characters appear in an environment where their Ukrainian behaviour is not wanted they should give an ‘expected’ result. Therefore we define shorthands at system level as well.

```
\declare@shorthand{system}{:}{\string:}
\declare@shorthand{system}{;}{\string;}
⟨/frenchpunct⟩
⟨*frenchpunct&!spanishligs⟩
\declare@shorthand{system}{!}{\string!}
\declare@shorthand{system}{?}{\string?}
⟨/frenchpunct&!spanishligs⟩
```

To be able to define the function of '"', we first define a couple of 'support' macros.

`\dq` We save the original double quote character in `\dq` to keep it available, the math accent `\"` can now be typed as '"'.

```
\begingroup \catcode`\"12
\def\reserved@a{\endgroup
  \def\@SS{\mathchar"7019 }
  \def\dq{"}}
\reserved@a
```

Now we can define the doublequote macros: german and french quotes. We use definitions of these quotes made in babel.sty. The french quotes are contained in the `T2*` encodings.

```
\declare@shorthand{ukrainian}{"`}{\glqq}
\declare@shorthand{ukrainian}{"'}{\grqq}
\declare@shorthand{ukrainian}{"<}{\flqq}
\declare@shorthand{ukrainian}{">}{\frqq}
```

Some additional commands:

```
\declare@shorthand{ukrainian}{""}{\hskip\z@skip}
\declare@shorthand{ukrainian}{"~}{\textormath{\leavevmode\hbox{-}}{-}}
\declare@shorthand{ukrainian}{"=}{\nobreak-\hskip\z@skip}
\declare@shorthand{ukrainian}{"|}{%
  \textormath{\nobreak\discretionary{-}{}{\kern.03em}%
              \allowhyphens}{}}
```

The next two macros for `"-` and `"---` are somewhat different. We must check whether the second token is a hyphen character:

```
\declare@shorthand{ukrainian}{"-}{%
```

If the next token is '`-`', we typeset an emdash, otherwise a hyphen sign:

```
  \def\ukrainian@sh@tmp{%
    \if\ukrainian@sh@next-\expandafter\ukrainian@sh@emdash
    \else\expandafter\ukrainian@sh@hyphen\fi
  }%
```

TeX looks for the next token after the first '`-`': the meaning of this token is written to `\ukrainian@sh@next` and `\ukrainian@sh@tmp` is called.

```
  \futurelet\ukrainian@sh@next\ukrainian@sh@tmp}
```

Here are the definitions of hyphen and emdash. First the hyphen:

```
\def\ukrainian@sh@hyphen{%
  \nobreak\-\bbl@allowhyphens}
```

For the emdash definition, there are the two parameters: we must 'eat' two last hyphen signs of our emdash. . . :

```
\def\ukrainian@sh@emdash#1#2{\cdash-#1#2}
```

`\cdash` … these two parameters are useful for another macro: `\cdash`:

```
%\ifx\cdash\undefined % should be defined earlier
\def\cdash#1#2#3{\def\tempx@{#3}%
\def\tempa@{-}\def\tempb@{~}\def\tempc@{*}%
 \ifx\tempx@\tempa@\@Acdash\else
  \ifx\tempx@\tempb@\@Bcdash\else
   \ifx\tempx@\tempc@\@Ccdash\else
    \errmessage{Wrong usage of cdash}\fi\fi\fi}
```

second parameter (or third for `\cdash`) shows what kind of emdash to create in next step

`"---` ordinary (plain) Cyrillic emdash inside text: an unbreakable thinspace will be inserted before only in case of a *space* before the dash (it is necessary for dashes after display maths formulae: there could be lists, enumerations etc. started with "— where $a$ is ..." i.e., the dash starts a line). (Firstly there were planned rather soft rules for user: he may put a space before the dash or not. But it is difficult to place this thinspace automatically, i.e., by checking modes because after display formulae TEX uses horizontal mode. Maybe there is a misunderstanding? Maybe there is another way?) After a dash a breakable thinspace is always placed;

```
% What is more grammatically: .2em or .2\fontdimen6\font ?
\def\@Acdash{\ifdim\lastskip>\z@\unskip\nobreak\hskip.2em\fi
  \cyrdash\hskip.2em\ignorespaces}%
```

`"--~` emdash in compound names or surnames (like Mendeleev–Klapeiron); this dash has no space characters around; after the dash some space is added `\exhyphenalty`

```
\def\@Bcdash{\leavevmode\ifdim\lastskip>\z@\unskip\fi
 \nobreak\cyrdash\penalty\exhyphenpenalty\hskip\z@skip\ignorespaces}%
```

`"--*` for denoting direct speech (a space like `\enskip` must follow the emdash);

```
\def\@Ccdash{\leavevmode
 \nobreak\cyrdash\nobreak\hskip.35em\ignorespaces}%
%\fi
```

`\cyrdash` Finally the macro for "body" of the Cyrillic emdash. The `\cyrdash` macro will be defined in case this macro hasn't been defined in a fontenc file. For T2* fonts, cyrdash will be placed in the code of the English emdash thus it uses ligature `---`.

```
% Is there an IF necessary?
\ifx\cyrdash\undefined
  \def\cyrdash{\hbox to.8em{--\hss--}}
\fi
```

Here a really new macro—to place thinspace between initials. This macro used instead of `\,` allows hyphenation in the following surname.

```
%\declare@shorthand{ukrainian}{",}{\nobreak\hskip.2em\ignorespaces}
```

`\mdqon` `\mdqoff` All that's left to do now is to define a couple of commands for `"`.

```
\def\mdqon{\bbl@activate{"}}
\def\mdqoff{\bbl@deactivate{"}}
```

The Ukrainian hyphenation patterns can be used with `\lefthyphenmin` and `\righthyphenmin` set to 2.

```
\providehyphenmins{\CurrentOption}{\tw@\tw@}
% temporary hack:
\ifx\englishhyphenmins\undefined
  \def\englishhyphenmins{\tw@\thr@@}
\fi
```

Now the action `\extrasukrainian` has to execute is to make sure that the command `\frenchspacing` is in effect. If this is not the case the execution of `\noextrasukrainian` will switch it off again.

```
\addto\extrasukrainian{\bbl@frenchspacing}
\addto\noextrasukrainian{\bbl@nonfrenchspacing}
```

Next we add a new enumeration style for Ukrainian manuscripts with Cyrillic letters, and later on we define some math operator names in accordance with Ukrainian and Russian typesetting traditions.

`\Asbuk` We begin by defining `\Asbuk` which works like `\Alph`, but produces (uppercase) Cyrillic letters intead of Latin ones. The letters CYRGUP, and SFTSN are skipped, as usual for such enumeration.

```
\def\Asbuk#1{\expandafter\@Asbuk\csname c@#1\endcsname}
\def\@Asbuk#1{\ifcase#1\or
  \CYRA\or\CYRB\or\CYRV\or\CYRG\or\CYRD\or\CYRE\or\CYRIE\or
  \CYRZH\or\CYRZ\or\CYRI\or\CYRII\or\CYRYI\or\CYRISHRT\or
  \CYRK\or\CYRL\or\CYRM\or\CYRN\or\CYRO\or\CYRP\or\CYRR\or
  \CYRS\or\CYRT\or\CYRU\or\CYRF\or\CYRH\or\CYRC\or\CYRCH\or
  \CYRSH\or\CYRSHCH\or\CYRYU\or\CYRYA\else\@ctrerr\fi}
```

`\asbuk` The macro `\asbuk` is similar to `\alph`; it produces lowercase Ukrainian letters.

```
\def\asbuk#1{\expandafter\@asbuk\csname c@#1\endcsname}
\def\@asbuk#1{\ifcase#1\or
  \cyra\or\cyrb\or\cyrv\or\cyrg\or\cyrd\or\cyre\or\cyrie\or
  \cyrzh\or\cyrz\or\cyri\or\cyrii\or\cyryi\or\cyrishrt\or
  \cyrk\or\cyrl\or\cyrm\or\cyrn\or\cyro\or\cyrp\or\cyrr\or
  \cyrs\or\cyrt\or\cyru\or\cyrf\or\cyrh\or\cyrc\or\cyrch\or
  \cyrsh\or\cyrshch\or\cyryu\or\cyrya\else\@ctrerr\fi}
```

Set up default Cyrillic math alphabets. The math groups for cyrillic letters are defined in the encoding definition files. First, declare a new alphabet for symbols, `\cyrmathrm`, based on the symbol font for Cyrillic letters defined in the encoding definition file. Note, that by default Cyrillic letters are taken from upright font in math mode (unlike Latin letters).

```
%\RequirePackage{textmath}
```

```
\@ifundefined{sym\cyrillicencoding letters}{}{%
\SetSymbolFont{\cyrillicencoding letters}{bold}\cyrillicencoding
  \rmdefault\bfdefault\updefault
\DeclareSymbolFontAlphabet\cyrmathrm{\cyrillicencoding letters}
```

And we need a few commands to be able to switch to different variants.

```
\DeclareMathAlphabet\cyrmathbf\cyrillicencoding
  \rmdefault\bfdefault\updefault
\DeclareMathAlphabet\cyrmathsf\cyrillicencoding
  \sfdefault\mddefault\updefault
\DeclareMathAlphabet\cyrmathit\cyrillicencoding
  \rmdefault\mddefault\itdefault
\DeclareMathAlphabet\cyrmathtt\cyrillicencoding
  \ttdefault\mddefault\updefault
%
\SetMathAlphabet\cyrmathsf{bold}\cyrillicencoding
  \sfdefault\bfdefault\updefault
\SetMathAlphabet\cyrmathit{bold}\cyrillicencoding
  \rmdefault\bfdefault\itdefault
}
```

Some math functions in Ukrainian and Russian math books have other names: e.g., `sinh` in Russian is written as `sh` etc. So we define a number of new math operators.

`\sinh`:

```
\def\sh{\mathop{\operator@font sh}\nolimits}
```

`\cosh`:

```
\def\ch{\mathop{\operator@font ch}\nolimits}
```

`\tan`:

```
\def\tg{\mathop{\operator@font tg}\nolimits}
```

`\arctan`:

```
\def\arctg{\mathop{\operator@font arctg}\nolimits}
```

arcctg:

```
\def\arcctg{\mathop{\operator@font arcctg}\nolimits}
```

The following macro conflicts with `\th` defined in Latin 1 encoding:

`\tanh`:

```
\addto\extrasrussian{%
  \babel@save{\th}%
  \let\ltx@th\th
  \def\th{\textormath{\ltx@th}%
                      {\mathop{\operator@font th}\nolimits}}%
  }
```

`\cot`:

```
\def\ctg{\mathop{\operator@font ctg}\nolimits}
```

`\coth`:

```
\def\cth{\mathop{\operator@font cth}\nolimits}
```

\csc:

```
\def\cosec{\mathop{\operator@font cosec}\nolimits}
```

And finally some other Ukrainian and Russian mathematical symbols:

```
\def\Prob{\mathop{\kern\z@\mathsf{P}}\nolimits}
\def\Variance{\mathop{\kern\z@\mathsf{D}}\nolimits}
\def\nsd{\mathop{\cyrmathrm{\cyrn.\cyrs.\cyrd.}}\nolimits}
\def\nsk{\mathop{\cyrmathrm{\cyrn.\cyrs.\cyrk.}}\nolimits}
\def\NSD{\mathop{\cyrmathrm{\CYRN\CYRS\CYRD}}\nolimits}
\def\NSK{\mathop{\cyrmathrm{\CYRN\CYRS\CYRK}}\nolimits}
  \def\nod{\mathop{\cyrmathrm{\cyrn.\cyro.\cyrd.}}\nolimits}    % ??????
  \def\nok{\mathop{\cyrmathrm{\cyrn.\cyro.\cyrk.}}\nolimits}    % ??????
  \def\NOD{\mathop{\cyrmathrm{\CYRN\CYRO\CYRD}}\nolimits}       % ??????
  \def\NOK{\mathop{\cyrmathrm{\CYRN\CYRO\CYRK}}\nolimits}       % ??????
\def\Proj{\mathop{\cyrmathrm{\CYRP\cyrr}}\nolimits}
```

This is for compatibility with older Ukrainian packages.

```
\DeclareRobustCommand{\No}{%
   \ifmmode{\nfss@text{\textnumero}}\else\textnumero\fi}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{ukrainian}
⟨/code⟩
```

## 57 The Lower Sorbian language

The file `lsorbian.dtx`[65] It defines all the language-specific macros for Lower Sorbian.

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
⟨*code⟩
\LdfInit{lsorbian}\captionslsorbian
```

When this file is read as an option, i.e. by the `\usepackage` command, `lsorbian` will be an 'unknown' languagein which case we have to make it known. So we check for the existence of `\l@lsorbian` to see whether we have to do something here.

```
\ifx\l@lsorbian\@undefined
  \@nopatterns{Lsorbian}
  \adddialect\l@lsorbian\l@usorbian\fi
```

The next step consists of defining commands to switch to (and from) the Lower Sorbian language.

`\captionslsorbian` The macro `\captionslsorbian` defines all strings used in the four standard documentclasses provided with LaTeX.

```
\addto\captionslsorbian{%
  \def\prefacename{Zawod}%
  \def\refname{Referency}%
  \def\abstractname{Abstrakt}%
  \def\bibname{Literatura}%
  \def\chaptername{Kapitl}%
  \def\appendixname{Dodawki}%
  \def\contentsname{Wop\'simje\'se}%
  \def\listfigurename{Zapis wobrazow}%
  \def\listtablename{Zapis tabulkow}%
  \def\indexname{Indeks}%
  \def\figurename{Wobraz}%
  \def\tablename{Tabulka}%
  \def\partname{\'Z\v el}%
  \def\enclname{P\'si\l oga}%
  \def\ccname{CC}%
  \def\headtoname{Komu}%
  \def\pagename{Strona}%
  \def\seename{gl.}%
  \def\alsoname{gl.~teke}%
  \def\proofname{Proof}%  <-- needs translation
  \def\glossaryname{Glossary}% <-- Needs translation
  }%
```

[65] The file described in this section has version number v1.0f and was last revised on 2005/03/31. It was written by Eduard Werner (`edi@kaihh.hanse.de`).

`\newdatelsorbian` The macro `\newdatelsorbian` redefines the command `\today` to produce Lower Sorbian dates.

```
\def\newdatelsorbian{%
  \def\today{\number\day.~\ifcase\month\or
    januara\or februara\or m\v erca\or apryla\or maja\or
    junija\or julija\or awgusta\or septembra\or oktobra\or
    nowembra\or decembra\fi
    \space \number\year}}
```

`\olddatelsorbian` The macro `\olddatelsorbian` redefines the command `\today` to produce old-style Lower Sorbian dates.

```
\def\olddatelsorbian{%
  \def\today{\number\day.~\ifcase\month\or
    wjelikego ro\v zka\or
    ma\l ego ro\v zka\or
    nal\v etnika\or
    jat\v sownika\or
    ro\v zownika\or
    sma\v znika\or
    pra\v znika\or
    \v znje\'nca\or
    po\v znje\'nca\or
    winowca\or
    nazymnika\or
    godownika\fi \space \number\year}}
```

The default will be the new-style dates.

```
\let\datelsorbian\newdatelsorbian
```

`\extraslsorbian` `\noextraslsorbian` The macro `\extraslsorbian` will perform all the extra definitions needed for the lsorbian language. The macro `\noextraslsorbian` is used to cancel the actions of `\extraslsorbian`. For the moment these macros are empty but they are defined for compatibility with the other language definition files.

```
\addto\extraslsorbian{}
\addto\noextraslsorbian{}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{lsorbian}
⟨/code⟩
```

## 58 The Upper Sorbian language

The file `usorbian.dtx`[66] It defines all the language-specific macros for Upper Sorbian.

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
⟨*code⟩
\LdfInit{usorbian}\captionsusorbian
```

When this file is read as an option, i.e. by the `\usepackage` command, `usorbian` will be an 'unknown' languagein which case we have to make it known. So we check for the existence of `\l@usorbian` to see whether we have to do something here.

```
\ifx\l@usorbian\@undefined
    \@nopatterns{Usorbian}
    \adddialect\l@usorbian0\fi
```

The next step consists of defining commands to switch to (and from) the Upper Sorbian language.

`\captionsusorbian` The macro `\captionsusorbian` defines all strings used in the four standard documentclasses provided with LaTeX.

```
\addto\captionsusorbian{%
  \def\prefacename{Zawod}%
  \def\refname{Referency}%
  \def\abstractname{Abstrakt}%
  \def\bibname{Literatura}%
  \def\chaptername{Kapitl}%
  \def\appendixname{Dodawki}%
  \def\contentsname{Wobsah}%
  \def\listfigurename{Zapis wobrazow}%
  \def\listtablename{Zapis tabulkow}%
  \def\indexname{Indeks}%
  \def\figurename{Wobraz}%
  \def\tablename{Tabulka}%
  \def\partname{D\'z\v el}%
  \def\enclname{P\v r\l oha}%
  \def\ccname{CC}%
  \def\headtoname{Komu}%
  \def\pagename{Strona}%
  \def\seename{hl.}%
  \def\alsoname{hl.~te\v z}
  \def\proofname{Proof}%  <-- needs translation
  \def\glossaryname{Glossary}% <-- Needs translation
  }%
```

[66] The file described in this section has version number v1.0i and was last revised on 2005/03/31. It was written by Eduard Werner (`edi@kaihh.hanse.de`).

\newdateusorbian The macro \newdateusorbian redefines the command \today to produce Upper Sorbian dates.

```
\def\newdateusorbian{%
  \def\today{\number\day.~\ifcase\month\or
    januara\or februara\or m\v erca\or apryla\or meje\or junija\or
    julija\or awgusta\or septembra\or oktobra\or
    nowembra\or decembra\fi
    \space \number\year}}
```

\olddateusorbian The macro \olddateusorbian redefines the command \today to produce old-style Upper Sorbian dates.

```
\def\olddateusorbian{%
  \def\today{\number\day.~\ifcase\month\or
    wulkeho r\'o\v zka\or ma\l eho r\'o\v zka\or nal\v etnika\or
    jutrownika\or r\'o\v zownika\or  sma\v znika\or pra\v znika\or
    \v znjenca\or po\v znjenca\or winowca\or nazymnika\or
    hodownika\fi \space \number\year}}
```

The default will be the new-style dates.

```
\let\dateusorbian\newdateusorbian
```

\extrasusorbian The macro \extrasusorbian will perform all the extra definitions needed for the Upper Sorbian language. It's pirated from germanb.sty. The macro \noextrasusorbian is used to cancel the actions of \extrasusorbian.

Because for Upper Sorbian (as well as for Dutch) the " character is made active. This is done once, later on its definition may vary.

```
\initiate@active@char{"}
\addto\extrasusorbian{\languageshorthands{usorbian}}
\addto\extrasusorbian{\bbl@activate{"}}
```

Don't forget to turn the shorthands off again.

```
\addto\noextrasusorbian{\bbl@deactivate{"}}
```

In order for TEX to be able to hyphenate German Upper Sorbian words which contain 'ß' we have to give the character a nonzero \lccode (see Appendix H, the TEXbook).

```
\addto\extrasusorbian{\babel@savevariable{\lccode`\^^Y}%
  \lccode`\^^Y`\^^Y}
```

The umlaut accent macro \" is changed to lower the umlaut dots. The redefinition is done with the help of \umlautlow.

```
\addto\extrasusorbian{\babel@save\"\umlautlow}
\addto\noextrasusorbian{\umlauthigh}
```

The Upper Sorbian hyphenation patterns can be used with \lefthyphenmin and \righthyphenmin set to 2.

```
\providehyphenmins{\CurrentOption}{\tw@\tw@}
```

`\dq` We save the original double quote character in `\dq` to keep it available, the math accent `\"` can now be typed as `"`. Also we store the original meaning of the command `\"` for future use.

```
\begingroup \catcode`\"12
\def\x{\endgroup
  \def\@SS{\mathchar"7019 }
  \def\dq{"}}
\x
```

Now we can define the doublequote macros: the umlauts,

```
\declare@shorthand{usorbian}{"a}{\textormath{\"{a}}{\ddot a}}
\declare@shorthand{usorbian}{"o}{\textormath{\"{o}}{\ddot o}}
\declare@shorthand{usorbian}{"u}{\textormath{\"{u}}{\ddot u}}
\declare@shorthand{usorbian}{"A}{\textormath{\"{A}}{\ddot A}}
\declare@shorthand{usorbian}{"O}{\textormath{\"{O}}{\ddot O}}
\declare@shorthand{usorbian}{"U}{\textormath{\"{U}}{\ddot U}}
```

tremas,

```
\declare@shorthand{usorbian}{"e}{\textormath{\"{e}}{\ddot e}}
\declare@shorthand{usorbian}{"E}{\textormath{\"{E}}{\ddot E}}
\declare@shorthand{usorbian}{"i}{\textormath{\"{\i}}{\ddot\imath}}
\declare@shorthand{usorbian}{"I}{\textormath{\"{I}}{\ddot I}}
```

usorbian es-zet (sharp s),

```
\declare@shorthand{usorbian}{"s}{\textormath{\ss{}}{\@SS{}}}
\declare@shorthand{usorbian}{"S}{SS}
```

german and french quotes,

```
\declare@shorthand{usorbian}{"`}{%
  \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
\declare@shorthand{usorbian}{"'}{%
  \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
\declare@shorthand{usorbian}{"<}{%
  \textormath{\guillemotleft}{\mbox{\guillemotleft}}}
\declare@shorthand{usorbian}{">}{%
  \textormath{\guillemotright}{\mbox{\guillemotright}}}
```

discretionary commands

```
\declare@shorthand{usorbian}{"c}{\textormath{\bbl@disc ck}{c}}
\declare@shorthand{usorbian}{"C}{\textormath{\bbl@disc CK}{C}}
\declare@shorthand{usorbian}{"f}{\textormath{\bbl@disc f{ff}}{f}}
\declare@shorthand{usorbian}{"F}{\textormath{\bbl@disc F{FF}}{F}}
\declare@shorthand{usorbian}{"l}{\textormath{\bbl@disc l{ll}}{l}}
\declare@shorthand{usorbian}{"L}{\textormath{\bbl@disc L{LL}}{L}}
\declare@shorthand{usorbian}{"m}{\textormath{\bbl@disc m{mm}}{m}}
\declare@shorthand{usorbian}{"M}{\textormath{\bbl@disc M{MM}}{M}}
\declare@shorthand{usorbian}{"n}{\textormath{\bbl@disc n{nn}}{n}}
\declare@shorthand{usorbian}{"N}{\textormath{\bbl@disc N{NN}}{N}}
\declare@shorthand{usorbian}{"p}{\textormath{\bbl@disc p{pp}}{p}}
\declare@shorthand{usorbian}{"P}{\textormath{\bbl@disc P{PP}}{P}}
\declare@shorthand{usorbian}{"t}{\textormath{\bbl@disc t{tt}}{t}}
```

```
\declare@shorthand{usorbian}{"T}{\textormath{\bbl@disc T{TT}}{T}}
```

and some additional commands:

```
\declare@shorthand{usorbian}{"-}{\nobreak\-\bbl@allowhyphens}
\declare@shorthand{usorbian}{"|}{%
  \textormath{\nobreak\discretionary{-}{}{\kern.03em}%
              \allowhyphens}{}}
\declare@shorthand{usorbian}{""}{\hskip\z@skip}
```

`\mdqon` `\mdqoff` `\ck` All that's left to do now is to define a couple of commands for reasons of compatibility with `german.sty`.

```
\def\mdqon{\shorthandon{"}}
\def\mdqoff{\shorthandoff{"}}
\def\ck{\allowhyphens\discretionary{k-}{k}{ck}\allowhyphens}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{usorbian}
⟨/code⟩
```

## 59 The Turkish language

The file `turkish.dtx`[67] defines all the language definition macros for the Turkish language[68].

Turkish typographic rules specify that a little 'white space' should be added before the characters '`:`', '`!`' and '='. In order to insert this white space automatically these characters are made 'active'. Also `\frenhspacing` is set.

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
⟨*code⟩
\LdfInit{turkish}\captionsturkish
```

When this file is read as an option, i.e. by the `\usepackage` command, `turkish` could be an 'unknown' language in which case we have to make it known. So we check for the existence of `\l@turkish` to see whether we have to do something here.

```
\ifx\l@turkish\@undefined
  \@nopatterns{Turkish}
  \adddialect\l@turkish0\fi
```

The next step consists of defining commands to switch to (and from) the Turkish language.

`\captionsturkish` The macro `\captionsturkish` defines all strings used in the four standard documentclasses provided with LaTeX.

```
\addto\captionsturkish{%
  \def\prefacename{\"Ons\"oz}%
  \def\refname{Kaynaklar}%
  \def\abstractname{\"Ozet}%
  \def\bibname{Kaynak\c ca}%
  \def\chaptername{B\"ol\"um}%
  \def\appendixname{Ek}%
  \def\contentsname{\.I\c cindekiler}%
  \def\listfigurename{\c Sekil Listesi}%
  \def\listtablename{Tablo Listesi}%
  \def\indexname{Dizin}%
  \def\figurename{\c Sekil}%
  \def\tablename{Tablo}%
  \def\partname{K\i s\i m}%
  \def\enclname{\.Ili\c sik}%
  \def\ccname{Di\u ger Al\i c\i lar}%
  \def\headtoname{Al\i c\i}%
  \def\pagename{Sayfa}%
  \def\subjectname{\.Ilgili}%
```

[67]The file described in this section has version number v1.2m and was last revised on 2005/03/31.

[68]Mustafa Burc, `z6001@rziris01.rrz.uni-hamburg.de` provided the code for this file. It is based on the work by Pierre Mackay; Turgut Uyar, `uyar@cs.itu.edu.tr` supplied additional translations in version 1.2j and later

```
  \def\seename{bkz.}%
  \def\alsoname{ayr\i ca bkz.}%
  \def\proofname{Kan\i t}%
  \def\glossaryname{Glossary}% <-- Needs translation
}%
```

`\dateturkish` The macro `\dateturkish` redefines the command `\today` to produce Turkish dates.

```
\def\dateturkish{%
  \def\today{\number\day~\ifcase\month\or
    Ocak\or \c Subat\or Mart\or Nisan\or May\i{}s\or Haziran\or
    Temmuz\or A\u gustos\or Eyl\"ul\or Ekim\or Kas\i{}m\or
    Aral\i{}k\fi
    \space\number\year}}
```

`\extrasturkish` `\noextrasturkish` The macro `\extrasturkish` will perform all the extra definitions needed for the Turkish language. The macro `\noextrasturkish` is used to cancel the actions of `\extrasturkish`.

Turkish typographic rules specify that a little 'white space' should be added before the characters '`:`', '`!`' and '`=`'. In order to insert this white space automatically these characters are made `\active`, so they have to be treated in a special way.

```
\initiate@active@char{:}
\initiate@active@char{!}
\initiate@active@char{=}
```

We specify that the turkish group of shorthands should be used.

```
\addto\extrasturkish{\languageshorthands{turkish}}
```

These characters are 'turned on' once, later their definition may vary.

```
\addto\extrasturkish{%
  \bbl@activate{:}\bbl@activate{!}\bbl@activate{=}}
```

For Turkish texts `\frenchspacing` should be in effect. We make sure this is the case and reset it if necessary.

```
\addto\extrasturkish{\bbl@frenchspacing}
\addto\noextrasturkish{\bbl@nonfrenchspacing}
```

`\turkish@sh@!@` `\turkish@sh@=@` `\turkish@sh@:@` The definitions for the three active characters were made using intermediate macros. These are defined now. The insertion of extra 'white space' should only happen outside math mode, hence the check `\ifmmode` in the macros.

```
\declare@shorthand{turkish}{:}{%
  \ifmmode
    \string:%
  \else\relax
    \ifhmode
      \ifdim\lastskip>\z@
        \unskip\penalty\@M\thinspace
      \fi
    \fi
    \string:%
  \fi}
\declare@shorthand{turkish}{!}{%
  \ifmmode
    \string!%
  \else\relax
    \ifhmode
      \ifdim\lastskip>\z@
        \unskip\penalty\@M\thinspace
      \fi
    \fi
    \string!%
  \fi}
\declare@shorthand{turkish}{=}{%
  \ifmmode
    \string=%
  \else\relax
    \ifhmode
      \ifdim\lastskip>\z@
        \unskip\kern\fontdimen2\font
        \kern-1.4\fontdimen3\font
      \fi
    \fi
    \string=%
  \fi}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{turkish}
⟨/code⟩
```

## 60 The Hebrew language

The file `hebrew.dtx`[69] provides the following packages and files for Hebrew language support:

`hebrew.ldf` file defines all the language-specific macros for the Hebrew language.

`rlbabel.def` file is used by `hebrew.ldf` for bidirectional versions of the major LaTeX commands and environments. It is designed to be used with other right-to-left languages, not only with Hebrew.

`hebcal.sty` package defines a set of macros for computing Hebrew date from Gregorian one.

Additional Hebrew input and font encoding definition files that should be included and used with `hebrew.ldf` are:

`hebinp.dtx` provides Hebrew input encodings, such as ISO 8859-8, MS Windows codepage 1255 or IBM PC codepage 862 (see Section 61 on page 400).

`hebrew.fdd` contains Hebrew font encodings, related font definition files and `hebfont` package that provides Hebrew font switching commands (see Section 62 on page 408 for further details).

LaTeX 2.09 compatibility files are included with `heb209.dtx` and gives possibility to compile existing LaTeX 2.09 Hebrew documents with small (if any) changes (see Section 63 on page 427 for details).

Finally, optional document class `hebtech` may be useful for writing theses and dissertations in both Hebrew and English (and any other languages included with babel). It designed to meet requirements of the Graduate School of the Technion — Israel Institute of Technology.

*As of version 2.3e hebtech is no longer distributed together with heblatex. It should be part of a new "hebclasses" package*

### 60.1 Acknowledgement

The following people have contributed to Hebrew package in one way or another, knowingly or unknowingly. In alphabetical order: Irina Abramovici, Yaniv Bar-gury, Yael Dubinsky, Sergio Fogel, Dan Haran, Rama Porrat, Michail Rozman, Alon Ziv.

Tatiana Samoilov and Vitaly Surazhsky found a number of serious bugs in preliminary version of Hebrew package.

A number of other people have contributed comments and information. Specific contributions are acknowledged within the document.

I want to thank my wife, Vita, and son, Mishka, for their infinite love and patience.

If you made a contribution and I haven't mentioned it, don't worry, it was an accident. I'm sorry. Just tell me and I will add you to the next version.

[69]The Hebrew language support files described in this section have version number v2.3h and were last revised on 2005/03/30.

## 60.2 The DOCSTRIP modules

The following modules are used in the implementation to direct DOCSTRIP in generating external files:

| | |
|---|---|
| driver | produce a documentation driver file |
| hebrew | produce Hebrew language support file |
| rightleft | create right-to-left support file |
| calendar | create Hebrew calendar package |

A typical DOCSTRIP command file would then have entries like:

```
\generateFile{hebrew.ldf}{t}{\from{hebrew.dtx}{hebrew}}
```

## 60.3 Hebrew language definitions

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
60.1 ⟨*hebrew⟩
60.2 \LdfInit{hebrew}{captionshebrew}
```

When this file is read as an option, i.e., by the `\usepackage` command, `hebrew` will be an 'unknown' language, in which case we have to make it known. So we check for the existence of `\l@hebrew` to see whether we have to do something here.

```
60.3 \ifx\l@hebrew\@undefined
60.4   \@nopatterns{Hebrew}%
60.5   \adddialect\l@hebrew0
60.6 \fi
```

`\hebrewencoding` *FIX DOCS REGARDING 8BIT*

Typesetting Hebrew texts implies that a special input and output encoding needs to be used. Generally, the user may choose between different available Hebrew encodings provided. The current support for Hebrew uses all available fonts from the Hebrew University of Jerusalem encoded in 'old-code' 7-bit encoding also known as Israeli Standard SI-960. We define for these fonts the Local Hebrew Encoding `LHE` (see the file `hebrew.fdd` for more details), and the `LHE` encoding definition file should be loaded by default.

Other fonts are available in windows-cp1255 (a superset of ISO-8859-8 with nikud). For those, the encoding `HE8` should be used. Such fonts are, e.g., windows' TrueType fonts (once cnverted to Type1 or MetaFont) and IBM's Type1 fonts.

However, if an user wants to use another font encoding, for example, cyrillic encoding T2 and extended latin encoding T1, — he/she has to load the corresponding file *before* the `hebrew` package. This may be done in the following way:

```
\usepackage[LHE,T2,T1]{fontenc}
\usepackage[hebrew,russian,english]{babel}
```

We make sure that the `LHE` encoding is known to LaTeX at end of this package.

Also note that if you want to use the encoding `HE8` , you should define the following in your document, *before loading babel*:

```
\def\HeblatexEncoding{HE8}
\def\HeblatexEncodingFile{he8enc}
```

```
\providecommand{\HeblatexEncoding}{LHE}%
\providecommand{\HeblatexEncodingFile}{lheenc}%
\newcommand{\heblatex@set@encoding}[2]{
}
\AtEndOfPackage{%
  \@ifpackageloaded{fontenc}{%
    \@ifl@aded{def}{%
      \HeblatexEncodingFile}{\def\hebrewencoding{\HeblatexEncoding}}{}%
  }{%
    \input{\HeblatexEncodingFile.def}%
    \def\hebrewencoding{\HeblatexEncoding}%
  }}
```

We also need to load inputenc package with one of the Hebrew input encodings. By default, we set up the `8859-8` codepage. If an user wants to use many input encodings in the same document, for example, the MS Windows Hebrew codepage `cp1255` and the standard IBM PC Russian codepage `cp866`, he/she has to load the corresponding file *before* the hebrew package too. This may be done in the following way:

```
\usepackage[cp1255,cp866]{inputenc}
\usepackage[hebrew,russian,english]{babel}
```

An user can switch input encodings in the document using the command `\inputencoding`, for example, to use the `cp1255`:

```
\inputencoding{cp1255}
```

```
\AtEndOfPackage{%
  \@ifpackageloaded{inputenc}{}{\RequirePackage[8859-8]{inputenc}}}
```

The next step consists of defining commands to switch to (and from) the Hebrew language.

`\hebrewhyphenmins` This macro is used to store the correct values of the hyphenation parameters `\lefthyphenmin` and `\righthyphenmin`. They are set to 2.

```
\providehyphenmins{\CurrentOption}{\tw@\tw@}
```

`\captionshebrew` The macro `\captionshebrew` replaces all captions used in the four standard document classes provided with LaTeX 2ε with their Hebrew equivalents.

```
\addto\captionshebrew{%
  \def\prefacename{\@ensure@R{\hebmem\hebbet\hebvav\hebalef}}%
  \def\refname{\@ensure@R{\hebresh\hebshin\hebyod\hebmem\hebtav\ %
    \hebmem\hebqof\hebvav\hebresh\hebvav\hebtav}}%
  \def\abstractname{\@ensure@R{\hebtav\hebqof\hebtsadi\hebyod\hebresh}}%
  \def\bibname{\@ensure@R{\hebbet\hebyod\hebbet\heblamed\hebyod\hebvav%
    \hebgimel\hebresh\hebpe\hebyod\hebhe}}%
  \def\chaptername{\@ensure@R{\hebpe\hebresh\hebqof}}%
  \def\appendixname{\@ensure@R{\hebnun\hebsamekh\hebpe\hebhet}}%
  \def\contentsname{\@ensure@R{%
    \hebtav\hebvav\hebkaf\hebfinalnun\ %
    \hebayin\hebnun\hebyod\hebyod\hebnun\hebyod\hebfinalmem}}%
  \def\listfigurename{\@ensure@R{%
    \hebresh\hebshin\hebyod\hebmem\hebtav\ %
    \hebalef\hebyod\hebvav\hebresh\hebyod\hebfinalmem}}%
  \def\listtablename{\@ensure@R{%
    \hebresh\hebshin\hebyod\hebmem\hebtav\
    \hebtet\hebbet\heblamed\hebalef\hebvav\hebtav}}%
  \def\indexname{\@ensure@R{\hebmem\hebpe\hebtav\hebhet}}%
  \def\figurename{\@ensure@R{\hebalef\hebyod\hebvav\hebresh}}%
  \def\tablename{\@ensure@R{\hebtet\hebbet\heblamed\hebhe}}%
  \def\partname{\@ensure@R{\hebhet\heblamed\hebqof}}%
  \def\enclname{\@ensure@R{\hebresh\hebtsadi"\hebbet}}%
  \def\ccname{\@ensure@R{\hebhe\hebayin\hebtav\hebqof\hebyod%
    \hebfinalmem}}%
  \def\headtoname{\@ensure@R{\hebalef\heblamed}}%
  \def\pagename{\@ensure@R{\hebayin\hebmem\hebvav\hebdalet}}%
  \def\psname{\@ensure@R{\hebnun.\hebbet.}}%
  \def\seename{\@ensure@R{\hebresh\hebalef\hebhe}}%
  \def\alsoname{\@ensure@R{\hebresh\hebalef\hebhe \hebgimel%
    \hebmemesof}}%
  \def\proofname{\@ensure@R{\hebhe\hebvav\hebkaf\hebhet\hebhe}}
  \def\glossaryname{\@ensure@L{Glossary}}% <-- Needs translation
}
```

`\slidelabel` Here we fix the macro `slidelabel` of the seminar package. Note that this still won't work well enough when overlays will be involved

```
\@ifclassloaded{seminar}{%
  \def\slidelabel{\bf \if@rl\R{\hebshin\hebqof\hebfinalpe{} \theslide}%
                      \else\L{Slide \theslide}%
                      \fi}%
}{}
```

Here we provide an user with translation of Gregorian dates to Hebrew. In addition, the `hebcal` package can be used to create Hebrew calendar dates.

`\hebmonth` The macro `\hebmonth{`*month*`}` produces month names in Hebrew.

```
\def\hebmonth#1{%
  \ifcase#1\or \hebyod\hebnun\hebvav\hebalef\hebresh\or %
    \hebpe\hebbet\hebresh\hebvav\hebalef\hebresh\or %
    \hebmem\hebresh\hebfinaltsadi\or %
    \hebalef\hebpe\hebresh\hebyod\heblamed\or %
    \hebmem\hebalef\hebyod\or \hebyod\hebvav\hebnun\hebyod\or %
    \hebyod\hebvav\heblamed\hebyod\or %
    \hebalef\hebvav\hebgimel\hebvav\hebsamekh\hebtet\or %
    \hebsamekh\hebpe\hebtet\hebmem\hebbet\hebresh\or %
    \hebalef\hebvav\hebqof\hebtet\hebvav\hebbet\hebresh\or %
```

```
      \hebnun\hebvav\hebbet\hebmem\hebbet\hebresh\or %
      \hebdalet\hebtsadi\hebmem\hebbet\hebresh\fi}
```

`\hebdate` The macro `\hebdate{`*day*`}{`*month*`}{`*year*`}` translates a given Gregorian date to Hebrew.

```
\def\hebdate#1#2#3{%
  \beginR\beginL\number#1\endL\ \hebbet\hebmonth{#2}
        \beginL\number#3\endL\endR}
```

`\hebday` The macro `\hebday` will replace `\today` command when in Hebrew mode.

```
\def\hebday{\hebdate{\day}{\month}{\year}}
```

`\datehebrew` The macro `\datehebrew` redefines the command `\today` to produce Gregorian dates in Hebrew. It uses the macro `\hebday`.

```
\def\datehebrew{\let\today=\hebday}
```

The macro `\extrashebrew` will perform all the extra definitions needed for the Hebrew language. The macro `\noextrashebrew` is used to cancel the actions of `\extrashebrew`.

`\extrashebrew` We switch font encoding to Hebrew and direction to right-to-left. We cannot use the regular language switching commands (for example, `\sethebrew` and `\unsethebrew` or `\selectlanguage{hebrew}`), when in restricted horizontal mode, because it will result in *unbalanced* `\beginR` or `\beginL` primitives. Instead, in TEX's restricted horizontal mode, the `\L{`*latin text*`}` and `\R{`*hebrew text*`}`, or `\embox{`*latin text*`}` and `\hmbox{`*hebrew text*`}` should be used.

Hence, we use `\beginR` and `\beginL` switching commands only when not in restricted horizontal mode.

```
\addto\extrashebrew{%
  \tohebrew%
  \ifhmode\ifinner\else\beginR\fi\fi}
```

`\noextrashebrew` The macro `\noextrashebrew` is used to cancel the actions of `\extrashebrew`. We switch back to the previous font encoding and restore left-to-right direction.

```
\addto\noextrashebrew{%
  \fromhebrew%
  \ifhmode\ifinner\else\beginL\fi\fi}
```

Generally, we can switch to- and from- Hebrew by means of standard babel-defined commands, for example,

```
\selectlanguage{hebrew}
```

or

```
\begin{otherlanguage}{hebrew}
   some Hebrew text
\end{otherlanguage}
```

Now we define two additional commands that offer the possibility to switch to and from Hebrew language. These commands are backward compatible with the previous versions of `hebrew.sty`.

`\sethebrew` `\unsethebrew` The command `\sethebrew` will switch from the current font encoding to the hebrew font encoding, and from the current direction of text to the right-to-left mode. The command `\unsethebrew` switches back.

Both commands use standard right-to-left switching macros `\setrllanguage{`*r2l language name*`}` and `\unsetrllanguage{`*r2l language name*`}`, that defined in the `rlbabel.def` file.

```
60.84 \def\sethebrew{\setrllanguage{hebrew}}
60.85 \def\unsethebrew{\unsetrllanguage{hebrew}}
```

`\hebrewtext` `\nohebrewtext` The following two commands are *obsolete* and work only in LaTeX2.09 compatibility mode. They are synonyms of `\sethebrew` and `\unsethebrew` defined above.

```
60.86 \if@compatibility
60.87   \let\hebrewtext=\sethebrew
60.88   \let\nohebrewtext=\unsethebrew
60.89 \fi
```

`\tohebrew` `\fromhebrew` These two commands change only the current font encoding to- and from- Hebrew encoding. Their implementation uses `\@torl{`*language name*`}` and `\@fromrl` macros defined in `rlbabel.def` file. Both commands may be useful *only* for package and class writers, not for regular users.

```
60.90 \def\tohebrew{\@torl{hebrew}}%
60.91 \def\fromhebrew{\@fromrl}
```

`\@hebrew` Sometimes we need to preserve Hebrew mode without knowing in which environment we are located now. For these cases, the `\@hebrew{`*hebrew text*`}` macro will be useful. Not that this macro is similar to the `\@number` and `\@latin` macros defined in `rlbabel.def` file.

```
60.92 \def\@@hebrew#1{\beginR{{\tohebrew#1}}\endR}
60.93 \def\@hebrew{\protect\@@hebrew}
```

### 60.3.1 Hebrew numerals

We provide commands to print numbers in the traditional notation using Hebrew letters. We need commands that print a Hebrew number from a decimal input, as well as commands to print the value of a counter as a Hebrew number.

`\if@gim@apost` `\if@gim@final` Hebrew numbers can be written in various styles: with or without apostrophes, and with the letters kaf, mem, nun, pe, tsadi as either final or initial forms when they are the last letters in the sequence. We provide two flags to set the style options.

```
60.94 \newif\if@gim@apost  % whether we print apostrophes
60.95 \newif\if@gim@final  % whether we use final or initial letters
```

`\hebrewnumeral` `\Hebrewnumeral` `\Hebrewnumeralfinal`

The commands that print a Hebrew number must specify the style locally: relying on a global style option could cause a counter to print in an inconsistent manner—for instance, page numbers might appear in different styles if the global style option changed mid-way through a document. The commands only allow three of the four possible flag combinations (I do not know of a use that requires the combination of final letters and no apostrophes –RA).

Each command sets the style flags and calls `\@hebrew@numeral`. Double braces are used in order to protect the values of `\@tempcnta` and `\@tempcntb`, which are changed by this call; they also keep the flag assignments local (this is not important because the global values are never used).

```
\newcommand*{\hebrewnumeral}[1]      % no apostrophe, no final letters
 {{\@gim@finalfalse\@gim@apostfalse\@hebrew@numeral{#1}}}
\newcommand*{\Hebrewnumeral}[1]      % apostrophe, no final letters
 {{\@gim@finalfalse\@gim@aposttrue\@hebrew@numeral{#1}}}
\newcommand*{\Hebrewnumeralfinal}[1] % apostrophe, final letters
 {{\@gim@finaltrue\@gim@aposttrue\@hebrew@numeral{#1}}}
```

`\alph` `\@alph` `\Alph` `\@Alph` `\Alphfinal` `\@Alphfinal`

Counter-printing commands are based on the above commands. The natural name for the counter-printing commands is `\alph`, because Hebrew numerals are the only way to represent numbers with Hebrew letters (kaf always means 20, never 11). Hebrew has no uppercase letters, hence no need for the familiar meaning of `\Alph`; we therefore define `\alph` to print counters as Hebrew numerals without apostrophes, and `\Alph` to print with apostrophes. A third form, `\Alphfinal`, is provided to print with apostrophes and final letters, as is required for Hebrew year designators. The commands `\alph` and `\Alph` are defined in `latex.ltx`, and we only need to redefine the internal commands `\@alph` and `\@Alph`; for `\Alphfinal` we need to provide both a wrapper and an internal command. The counter printing commands are made semi-robust: without the `\protect`, commands like `\theenumii` break (I'm not quite clear on why this happens, –RA); at the same time, we cannot make the commands too robust (e.g. with `\DeclareRobustCommand`) because this would enter the command name rather than its value into files like `.aux`, `.toc` etc. The old meanings of meaning of `\@alph` and `\@Alph` are saved upon entering Hebrew mode and restored upon exiting it.

```
\addto\extrashebrew{%
  \let\saved@alph=\@alph%
  \let\saved@Alph=\@Alph%
  \renewcommand*{\@alph}[1]{\protect\hebrewnumeral{\number#1}}%
  \renewcommand*{\@Alph}[1]{\protect\Hebrewnumeral{\number#1}}%
  \def\Alphfinal#1{\expandafter\@Alphfinal\csname c@#1\endcsname}%
  \providecommand*{\@Alphfinal}[1]{\protect\Hebrewnumeralfinal{\number#1}}}
\addto\noextrashebrew{%
  \let\@alph=\saved@alph%
  \let\@Alph=\saved@Alph}
```

Note that `\alph` (without apostrophes) is already the appropriate choice for the second-level enumerate label, and `\Alph` (with apostrophes) is an appropriate choice for appendix; however, the default LaTeX labels need to be redefined for appropriate cross-referencing, see below. LaTeX default class files specify `\Alph` for the fourth-level enumerate level, this should probably be changed. Also, the way labels get flushed left by default looks inappropriate for Hebrew numerals, so we should redefine `\labelenumii` as well as `\labelenumiv` (presently not implemented).

`\theenumii` `\theenumiv` `\label` Cross-references to counter labels need to be printed according to the language environment in which a label was issued, not the environment in which it is called: for example, a label (1b) issued in a Latin environment should be referred to as (1b) in a Hebrew text, and label (2dalet) issued in a Hebrew environment should be referred to as (2dalet) in a Latin text. This was the unanimous opinion in a poll sent to the IvriTEX list. We therefore redefine `\theenumii` and `\theenumiv`, so that an explicit language instruction gets written to the `.aux` file.

```
\renewcommand{\theenumii}
   {\if@rl\protect\hebrewnumeral{\number\c@enumii}%
    \else\protect\L{\protect\@@alph{\number\c@enumii}}\fi}
\renewcommand{\theenumiv}
   {\if@rl\protect\Hebrewnumeral{\number\c@enumiv}%
    \else\protect\L{\protect\@@Alph{\number\c@enumiv}}\fi}
```

We also need to control for the font and direction in which a counter label is printed. Direction is straightforward: a Latin label like (1b) should be written left-to-right when called in a Hebrew text, and a Hebrew label like (2dalet) should be written right-to-left when called in a Latin text. The font question is more delicate, because we should decide whether the numerals should be typeset in the font of the language enviroment in which the label was issued, or that of the environment in which it is called.

- A purely numeric label like (23) looks best if it is set in the font of the surrounding language.
- But a mixed alphanumeric label like (1b) lookes weird if the '1' is taken from the Hebrew font; likewise, (2dalet) looks weird if the '2' is taken from a Latin font.
- Finally, mixing the two possibilities is worst, because a single Hebrew sentence referring to examples (1b) and (2) would take the '1' from the Latin font and the '2' from the Hebrew font, and this looks really awful. (It is also very hard to implement).

In light of the conflicting considerations it seems like there's no perfect solution. I have chosen to implement the top option, where numerals are taken from the font of the surrounding language, because it seems to me that reference to purely numeric labels is the most common, so this gives a good solution to the majority of cases and a mediocre solution to the minority.

We redefine the `\label` command which writes to the `.aux` file. Depending on the language environment we issue appropriate `\beginR/L`···`\endR/L` commands to control the direction without affecting the font. Since these commands do not affect the value of `\if@rl`, we cannot use the macro `\@brackets` to determine the correct brackets to be used with `\p@enumiii`; instead, we let the language environment determine an explicit definition.

```
\def\label#1{\@bsphack
  \if@rl
    \def\p@enumiii{\p@enumii)\theenumii(}%
    \protected@write\@auxout{}%
         {\string\newlabel{#1}{{\beginR\@currentlabel\endR}{\thepage}}}%
  \else
    \def\p@enumiii{\p@enumii(\theenumii)}%
    \protected@write\@auxout{}%
         {\string\newlabel{#1}{{\beginL\@currentlabel\endL}{\thepage}}}%
  \fi
  \@esphack}
```

NOTE: it appears that the definition of `\label` is language-independent and thus belongs in `rlbabel.def`, but this is not the case. The decision to typeset label numerals in the font of the surrounding language is reasonable for Hebrew, because mixed-font (1b) and (2dalet) are somewhat acceptable. The same may not be acceptable for Arabic, whose numeral glyphs are radically different from those in the Latin fonts. The decision about the direction may also be different for Arabic, which is more right-to-left oriented than Hebrew (two examples: dates like 15/6/2003 are written left-to-right in Hebrew but right-to-left in Arabic; equations like $1 + 2 = 3$ are written left-to-right in Hebrew but right-to-left in Arabic elementary school textbooks using Arabic numeral glyphs). My personal hunch is that a label like (1b) in an Arabic text would be typeset left-to-right if the '1' is a Western glyph, but right-to-left if the '1' is an Arabic glyph. But this is just a guess, I'd have to ask Arab typesetters to find the correct answer. –RA.

`\appendix` The following code provides for the proper printing of appendix numbers in tables of contents. Section and chapter headings are normally bilingual: regardless of the text language, the author supplies each section/chapter with two headings—one for the Hebrew table of contents and one for the Latin table of contents. It makes sense that the label should be a Latin letter in the Latin table of contents and a Hebrew letter in the Hebrew table of contents. The definition is similar to that of `\theenumii` and `\theenumiv` above, but additional `\protect` commands ensure that the entire condition is written the `.aux` file. The appendix number will therefore be typeset according to the environment in which it is used rather than issued: a Hebrew number (with apostrophes) in a Hebrew environment and a Latin capital letter in a Latin environment (the command `\@@Alph` is set in `rlbabel.def` to hold the default meaning of LaTeX [latin] `\@Alph`, regardless of the mode in which it is issued). The net result is that the second appendix will be marked with 'B' in the Latin table of contents and with 'bet' in the Hebrew table of contents; the mark in the main text will depend on the language of the appendix itself.

```
\@ifclassloaded{letter}{}{%
\@ifclassloaded{slides}{}{%
  \let\@@appendix=\appendix%
  \@ifclassloaded{article}{%
```

```
   \renewcommand\appendix{\@@appendix%
     \renewcommand\thesection
       {\protect\if@rl\protect\Hebrewnumeral{\number\c@section}%
        \protect\else\@@Alph\c@section\protect\fi}}}
  {\renewcommand\appendix{\@@appendix%
     \renewcommand\thechapter
       {\protect\if@rl\protect\Hebrewnumeral{\number\c@chapter}%
        \protect\else\@@Alph\c@chapter\protect\fi}}}}
```

QUESTION: is this also the appropriate way to refer to an appendix in the text, or should we retain the original label the same way we did with `enumerate` labels? ANOTHER QUESTION: are similar redefinitions needed for other counters that generate texts in bilingual lists like `.lof`/`.fol` and `.lot`/`.tol`? –RA.

`\@hebrew@numeral` The command `\@hebrew@numeral` prints a Hebrew number. The groups of thousands, millions, billions are separated by apostrophes and typeset without apostrophes or final letters; the remainder (under 1000) is typeset conventionally, with the selected styles for apostrophes and final letters. The function calls on `\gim@no@mil` to typeset each three-digit block. The algorithm is recursive, but the maximum recursion depth is 4 because TEX only allows numbers up to $2^{31} - 1 = 2{,}147{,}483{,}647$. The typesetting routine is wrapped in `\@hebrew` in order to ensure that numbers are always typeset in Hebrew mode.

Initialize: `\@tempcnta` holds the value, `\@tempcntb` is used for calculations.

```
\newcommand*{\@hebrew@numeral}[1]
{\@hebrew{\@tempcnta=#1\@tempcntb=#1\relax
 \divide\@tempcntb by 1000
```

If we're under 1000, call `\gim@nomil`

```
 \ifnum\@tempcntb=0\gim@nomil\@tempcnta\relax
```

If we're above 1000 then force no apostrophe and no final letter styles for the value above 1000, recur for the value above 1000, add an apostrophe, and call `\gim@nomil` for the remainder.

```
 \else{\@gim@apostfalse\@gim@finalfalse\@hebrew@numeral\@tempcntb}'%
      \multiply\@tempcntb by 1000\relax
      \advance\@tempcnta by -\@tempcntb\relax
      \gim@nomil\@tempcnta\relax
 \fi
}}
```

NOTE: is it the case that 15,000 and 16,000 are written as yod-he and yod-vav, rather than tet-vav and tet-zayin? This vaguely rings a bell, but I'm not certain. If this is the case, then the current behavior is incorrect and should be changed. –RA.

`\gim@nomil` The command `\gim@nomil` typesets an integer between 0 and 999 (for 0 it typesets nothing). The code has been modified from the old `hebcal.sty` (appropriate credits—Boris Lavva and Michail Rozman ?). `\@tempcnta` holds the total value that remains to be typeset. At each stage we find the highest valued letter that is less than or equal to `\@tempcnta`, and call on `\gim@print` to subtract this value and print the letter.

Initialize: `\@tempcnta` holds the value, there is no previous letter.

```
\newcommand*{\gim@nomil}[1]{\@tempcnta=#1\@gim@prevfalse
```

Find the hundreds digit.

```
  \@tempcntb=\@tempcnta\divide\@tempcntb by 100\relax % hundreds digit
  \ifcase\@tempcntb                     % print nothing if no hundreds
     \or\gim@print{100}{\hebqof}%
     \or\gim@print{200}{\hebresh}%
     \or\gim@print{300}{\hebshin}%
     \or\gim@print{400}{\hebtav}%
     \or\hebtav\@gim@prevtrue\gim@print{500}{\hebqof}%
     \or\hebtav\@gim@prevtrue\gim@print{600}{\hebresh}%
     \or\hebtav\@gim@prevtrue\gim@print{700}{\hebshin}%
     \or\hebtav\@gim@prevtrue\gim@print{800}{\hebtav}%
     \or\hebtav\@gim@prevtrue\hebtav\gim@print{900}{\hebqof}%
  \fi
```

Find the tens digit. The numbers 15 and 16 are traditionally printed as tet-vav $(9+6)$ and tet-zayin $(9+7)$ to avoid spelling the Lord's name.

```
  \@tempcntb=\@tempcnta\divide\@tempcntb by 10\relax     % tens digit
  \ifcase\@tempcntb                         % print nothing if no tens
      \or                                   % number between 10 and 19
              \ifnum\@tempcnta = 16 \gim@print {9}{\hebtet}% tet-zayin
         \else\ifnum\@tempcnta = 15 \gim@print {9}{\hebtet}% tet-vav
         \else                      \gim@print{10}{\hebyod}%
              \fi % \@tempcnta = 15
              \fi % \@tempcnta = 16
```

Initial or final forms are selected according to the current style option; `\gim@print` will force a non-final letter in non-final position by means of a local style change.

```
      \or\gim@print{20}{\if@gim@final\hebfinalkaf\else\hebkaf\fi}%
      \or\gim@print{30}{\heblamed}%
      \or\gim@print{40}{\if@gim@final\hebfinalmem\else\hebmem\fi}%
      \or\gim@print{50}{\if@gim@final\hebfinalnun\else\hebnun\fi}%
      \or\gim@print{60}{\hebsamekh}%
      \or\gim@print{70}{\hebayin}%
      \or\gim@print{80}{\if@gim@final\hebfinalpe\else\hebpe\fi}%
      \or\gim@print{90}{\if@gim@final\hebfinaltsadi\else\hebtsadi\fi}%
  \fi
```

Print the ones digit.

```
  \ifcase\@tempcnta                         % print nothing if no ones
      \or\gim@print{1}{\hebalef}%
      \or\gim@print{2}{\hebbet}%
      \or\gim@print{3}{\hebgimel}%
      \or\gim@print{4}{\hebdalet}%
      \or\gim@print{5}{\hebhe}%
      \or\gim@print{6}{\hebvav}%
      \or\gim@print{7}{\hebzayin}%
      \or\gim@print{8}{\hebhet}%
      \or\gim@print{9}{\hebtet}%
  \fi
}
```

`\gim@print` `\if@gim@prev` The actual printing routine typesets a digit with the appropriate apostrophes: if a number sequence consists of a single letter then it is followed by a single apostrophe, and if it consists of more than one letter then a double apostrophe is inserted before the last letter. We typeset the letters one at a time, keeping a flag that tells us if any previous letters had been typeset.

```
\newif\if@gim@prev % flag if a previous letter has been typeset
```

For each letter, we first subtract its value from the total. Then,

- if the result is zero then this is the last letter; we check the flag to see if this is the only letter and print it with the appropriate apostrophe;
- if the result is not zero then there remain additional letters to be typeset; we print without an apostrophe and set the 'previous letter' flag.

`\@tempcnta` holds the total value that remains to be typeset. We first deduct the letter's value from `\@tempcnta`, so `\@tempcnta` is zero if and only if this is the last letter.

```
\newcommand*{\gim@print}[2]{%   #2 is a letter, #1 is its value.
  \advance\@tempcnta by -#1\relax% deduct the value from the remainder
```

If this is the last letter, we print with the appropriate apostrophe (depending on the style option): if there is a preceding letter, print `"x` if the style calls for apostrophes, `x` if it doesn't; otherwise, this is the only letter: print `x'` if the style calls for apostrophes, `x` if it doesn't.

```
  \ifnum\@tempcnta=0% if this is the last letter
     \if@gim@prev\if@gim@apost"\fi#2%
     \else#2\if@gim@apost'\fi\fi%
```

If this is not the last letter: print a non-final form (by forcing a local style option) and set the 'previous letter' flag.

```
  \else{\@gim@finalfalse#2}\@gim@prevtrue\fi}
```

`\hebr` `\gim` The older Hebrew counter commands `\hebr` and `\gim` are retained in order to keep older documents from breaking. They are set to be equivalent to `\alph`, and their use is deprecated. Note that `\hebr` gives different results than it had in the past—it now typesets 11 as yod-alef rather than kaf.

```
\let\hebr=\alph
\let\gim=\alph
```

For backward compatibility with 'older' `hebrew.sty` packages, we define Hebrew equivalents of some useful LaTeX commands. Note, however, that 8-bit macros defined in Hebrew are no longer supported.

```
\def\hebcopy{\protect\R{\hebhe\hebayin\hebtav\hebqof}}
\def\hebincl{\protect\R{\hebresh\hebtsadi"\hebbet}}
```

```
\def\hebpage{\protect\R{\hebayin\hebmem\hebvav\hebdalet}}
\def\hebto{\protect\R{\hebayin\hebdalet}}
```

`\hadgesh` produce "poor man's bold" (heavy printout), when used with normal font glyphs. It is advisable to use bold font (for example, *Dead Sea*) instead of this macro.

```
\def\hadgesh#1{\leavevmode\setbox0=\hbox{#1}%
  \kern-.025em\copy0\kern-\wd0
  \kern.05em\copy0\kern-\wd0
  \kern-.025em\raise.0433em\box0 }
```

`\piska` and `\piskapiska` sometimes used in 'older' hebrew sources, and should not be used in LaTeX 2ε.

```
\if@compatibility
  \def\piska#1{\item{#1}\hangindent=-\hangindent}
  \def\piskapiska#1{\itemitem{#1}\hangindent=-\hangindent}
\fi
```

The following commands are simply synonyms for the standard ones, provided with LaTeX 2ε.

```
\let\makafgadol=\textendash
\let\makafanak=\textemdash
\let\geresh=\textquoteright
\let\opengeresh=\textquoteright
\let\closegeresh=\textquoteleft
\let\openquote=\textquotedblright
\let\closequote=\textquotedblleft
\let\leftquotation=\textquotedblright
\let\rightquotation=\textquotedblleft
```

We need to ensure that Hebrew is used as the default right-to-left language at `\begin{document}`. The mechanism of defining the `\@rllanguagename` is the same as in babel's `\languagename`: the last right-to-left language in the `\usepackage{babel}` line is set as the default right-to-left language at document beginning.

For example, the following code:

```
\usepackage[russian,hebrew,arabic,greek,english]{babel}
```

will set the Arabic language as the default right-to-left language and the English language as the default language. As a result, the commands `\L{}` and `\embox{}` will use English and `\R{}` and `\hmbox{}` will use Arabic by default. These defaults can be changed with the next `\sethebrew` or `\selectlanguage{`*language name*`}` command.

```
\AtBeginDocument{\def\@rllanguagename{hebrew}}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{hebrew}
⟨/hebrew⟩
```

## 60.4 Right to left support

This file `rlbabel.def` defines necessary bidirectional macro support for LaTeX 2ε. It is designed for use not only with Hebrew, but with any Right-to-Left languages, supported by babel. The macros provided in this file are language and encoding independent.

Right-to-left languages will use TeX extensions, namely TeX primitives `\beginL`, `\endL` and `\beginR`, `\endR`, currently implemented only in ε-TeX and in TeX--XeT.

If ε-TeX is used, we should switch it to the *enhanced* mode:

```
⟨*rightleft⟩
\ifx\TeXXeTstate\undefined\else%
   \TeXXeTstate=1
\fi
```

Note, that ε-TeX's format file should be created for *extended* mode. Mode can be checked by running ε-TeX on some TeX file, for example:

```
This is e-TeX, Version 3.14159-1.1 (Web2c 7.0)
entering extended mode
```

The second line should be `entering extended mode`.

We check if user uses Right-to-Left enabled engine instead of regular Knuth's TeX:

```
\ifx\beginL\@undefined%
   \newlinechar`\^^J
   \typeout{^^JTo avoid this error message,^^J%
     run TeX--XeT or e-TeX engine instead of regular TeX.^^J}
   \errmessage{Right-to-Left Support Error: use TeX--XeT or e-TeX
     engine}%
\fi
```

### 60.4.1 Switching from LR to RL mode and back

`\@torl` and `\@fromrl` are called each time the horizontal direction changes. They do all that is necessary besides changing the direction. Currently their task is to change the encoding information and mode (condition `\if@rl`). They should not normally be called by users: user-level macros, such as `\sethebrew` and `\unsethebrew`, as well as babel's `\selectlanguage` are defined in language-definition files and should be used to change default language (and direction).

Local direction changing commands (for small pieces of text): `\L{}`, `\R{}`, `\embox{}` and `\hmbox{}` are defined below in this file in language-independent manner.

`\if@rl` `rltrue` means that the main mode is currently Right-to-Left.
`rlfalse` means that the main mode is currently Left-to-Right.

```
\newif\if@rl
```

`\if@rlmain` This is the main direction of the document. Unlike `\if@rl` it is set once and never changes.

`rltrue` means that the document is Right-to-Left.
`rlfalse` means that the document is Left-to-Right.

Practically `\if@rlmain` is set according to the value of `\if@rl` in the beginning of the run.

```
\AtBeginDocument{% Here we set the main document direction
  \newif\if@rlmain%
  \if@rl% e.g: if the options to babel were [english,hebrew]
    \@rlmaintrue%
  \else%  e.g: if the options to babel were [hebrew,english]
    \@rlmainfalse%
  \fi%
}
```

`\@torl` Switches current direction to Right-to-Left: saves current Left-to-Right encoding in `\lr@encodingdefault`, sets required Right-to-Left language name in `\@rllanguagename` (similar to babel's `\languagename`) and changes derection.

The Right-to-Left language encoding should be defined in `.ldf` file as special macro created by concatenation of the language name and string `encoding`, for example, for Hebrew it will be `\hebrewencoding`.

```
\DeclareRobustCommand{\@torl}[1]{%
  \if@rl\else%
     \let\lr@encodingdefault=\encodingdefault%
  \fi%
  \def\@rllanguagename{#1}%
  \def\encodingdefault{\csname#1encoding\endcsname}%
  \fontencoding{\encodingdefault}%
  \selectfont%
  \@rltrue}
```

`\@fromrl` Opposite to `\@torl`, switches current direction to Left-to-Right: restores saved Left-to-Right encoding (`\lr@encodingdefault`) and changes direction.

```
\DeclareRobustCommand{\@fromrl}{%
  \if@rl%
     \let\encodingdefault=\lr@encodingdefault%
  \fi%
  \fontencoding{\encodingdefault}%
  \selectfont%
  \@rlfalse}
```

`\selectlanguage` This standard babel's macro should be redefined to support bidirectional tables. We divide `\selectlanguage` implementation to two parts, and the first part calls the second `\@@selectlanguage`.

```
\expandafter\def\csname selectlanguage \endcsname#1{%
  \edef\languagename{%
    \ifnum\escapechar=\expandafter`\string#1\@empty
    \else \string#1\@empty\fi}%
  \@@selectlanguage{\languagename}}
```

`\@@selectlanguage` This new internal macro redefines a final part of the standard babel's `\selectlanguage` implementation.

Standard LaTeX provides us with 3 tables: Table of Contents (`.toc`), List of Figures (`.lof`), and List of Tables (`.lot`). In multi-lingual texts mixing Left-to-Right languages with Right-to-Left ones, the use of various directions in one table results in very ugly output. Therefore, these 3 standard tables will be used now only for Left-to-Right languages, and we will add 3 Right-to-Left tables (their extensions are simply reversed ones): RL Table of Contents (`.cot`), RL List of Figures (`.fol`), and RL List of Tables (`.lof`).

```
\def\@@selectlanguage#1{%
  \select@language{#1}%
  \if@filesw
     \protected@write\@auxout{}{\string\select@language{#1}}%
     \if@rl%
       \addtocontents{cot}{\xstring\select@language{#1}}%
       \addtocontents{fol}{\xstring\select@language{#1}}%
       \addtocontents{tol}{\xstring\select@language{#1}}%
     \else%
       \addtocontents{toc}{\xstring\select@language{#1}}%
       \addtocontents{lof}{\xstring\select@language{#1}}%
       \addtocontents{lot}{\xstring\select@language{#1}}%
     \fi%
  \fi}
```

`\setrllanguage` `\unsetrllanguage` The `\setrllanguage` and `\unsetrllanguage` pair of macros is proved to very useful in bilingual texts, for example, in Hebrew-English texts. The language-specific commands, for example, `\sethebrew` and `\unsethebrew` use these macros as basis.

Implementation saves and restores other language in `\other@languagename` variable, and uses internal macro `\@@selectlanguage`, defined above, to switch between languages.

```
\let\other@languagename=\languagename
\DeclareRobustCommand{\setrllanguage}[1]{%
   \if@rl\else%
     \let\other@languagename=\languagename%
   \fi%
     \def\languagename{#1}%
     \@@selectlanguage{\languagename}}
```

```
\DeclareRobustCommand{\unsetrllanguage}[1]{%
   \if@rl%
     \let\languagename=\other@languagename%
   \fi
   \@@selectlanguage{\languagename}}
```

\L
\R
\HeblatexRedefineL

Macros for changing direction, originally taken from TUGboat. Usage: `\L{`*Left to Right text*`}` and `\R{`*Right to Left text*`}`. Numbers should also be enclosed in `\L{}`, as in `\L{123}`.

Note, that these macros do not receive language name as parameter. Instead, the saved `\@rllanguagename` will be used. We assume that each Right-to-Left language defines `\to`*languagename* and `\from`*languagename* macros in language definition file, for example, for Hebrew: `\tohebrew` and `\fromhebrew` macros in `hebrew.ldf` file.

The macros `\L` and `\R` include 'protect' to to make them robust and allow use, for example, in tables.

Due to the fact that some packages have different definitions for `\L` the macro `\HeblatexRedefineL` is provided to overide them. This may be required with hyperref, for instance.

```
60.293 \let\next=\
60.294 \def\HeblatexRedefineL{%
60.295   \def\L{\protect\pL}%
60.296 }
60.297 \HeblatexRedefineL
60.298 \def\pL{\protect\afterassignment\moreL \let\next= }
60.299 \def\moreL{\bracetext \aftergroup\endL \beginL\csname
60.300   from\@rllanguagename\endcsname}

60.301 \def\R{\protect\pR}
60.302 \def\pR{\protect\afterassignment\moreR \let\next= }
60.303 \def\moreR{\bracetext \aftergroup\endR \beginR\csname
60.304   to\@rllanguagename\endcsname}
60.305 \def\bracetext{\ifcat\next{\else\ifcat\next}\fi
60.306   \errmessage{Missing left brace has been substituted}\fi \bgroup}
60.307 \everydisplay{\if@rl\aftergroup\beginR\fi }
```

\@ensure@R
\@ensure@L

Two small internal macros, a-la `\ensuremath`

```
60.308 \def\@ensure@R#1{\if@rl#1\else\R{#1}\fi}
60.309 \def\@ensure@L#1{\if@rl\L{#1}\else#1\fi}
```

Take care of Right-to-Left indentation in every paragraph. Originally, `\noindent` was redefined for right-to-left by Yaniv Bargury, then the implementation was rewritten by Alon Ziv using an idea by Chris Rowley: `\noindent` now works unmodified.

```
60.310 \def\rl@everypar{\if@rl{\setbox\z@\lastbox\beginR\usebox\z@}\fi}
60.311 \let\o@everypar=\everypar
60.312 \def\everypar#1{\o@everypar{\rl@everypar#1}}
```

\hmbox
\embox

Useful vbox commands. All text in math formulas is best enclosed in these: LR text in `\embox` and RL text in `\hmbox`. `\mbox{}` is useless for both cases, since it typesets in Left-to-Right even for Right-to-Left languages (additions by Yaniv Bargury).

```
60.313 \newcommand{\hmbox}[1]{\mbox{\R{#1}}}
60.314 \newcommand{\embox}[1]{\mbox{\L{#1}}}
```

`\@brackets` When in Right-to-Left mode, brackets should be swapped. This macro receives 3 parameters: left bracket, content, right bracket. Brackets can be square brackets, braces, or parentheses.

```
\def\@brackets#1#2#3{\protect\if@rl #3#2#1\protect\else
  #1#2#3\protect\fi}
```

`\@number` `\@latin` `\@number` preserves numbers direction from Left to Right. `\@latin` in addition switches current encoding to the latin.

```
\def\@@number#1{\ifmmode\else\beginL\fi#1\ifmmode\else\endL\fi}
\def\@@latin#1{\@@number{{\@fromrl#1}}}
\def\@number{\protect\@@number}
\def\@latin{\protect\@@latin}
```

### 60.4.2 Counters

To make counter references work in Right to Left text, we need to surround their original definitions with an `\@number{...}` or `\@latin{...}`. Note, that language-specific counters, such as `\hebr` or `\gim` are provided with language definition file.

We start with saving the original definitions:

```
\let\@@arabic=\@arabic
\let\@@roman=\@roman
\let\@@Roman=\@Roman
\let\@@alph=\@alph
\let\@@Alph=\@Alph
```

`\@arabic` `\@roman` `\@Roman` Arabic and roman numbers should be from Left to Right. In addition, roman numerals, both lower- and upper-case should be in latin encoding.

```
\def\@arabic#1{\@number{\@@arabic#1}}
\def\@roman#1{\@latin{\@@roman#1}}
\def\@Roman#1{\@latin{\@@Roman#1}}
```

`\arabicnorl` This macro preserves the original definition of `\arabic` (overrides the overriding of `\@arabic`)

```
\def\arabicnorl#1{\expandafter\@@arabic\csname c@#1\endcsname}
```

`\make@lr` In Right to Left documents all counters defined in the standard document classes *article*, *report* and *book* provided with LaTeX $2_\varepsilon$, such as `\thesection`, `\thefigure`, `\theequation` should be typed as numbers from left to right. To ensure direction, we use the following `\make@lr{`*counter*`}` macro:

```
\def\make@lr#1{\begingroup
    \toks@=\expandafter{#1}%
    \edef\x{\endgroup
  \def\noexpand#1{\noexpand\@number{\the\toks@}}}%
  \x}
```

```
\@ifclassloaded{letter}{}{%
  \@ifclassloaded{slides}{}{%
    \make@lr\thesection
    \make@lr\thesubsection
    \make@lr\thesubsubsection
    \make@lr\theparagraph
    \make@lr\thesubparagraph
    \make@lr\thefigure
    \make@lr\thetable
  }
  \make@lr\theequation
}
```

### 60.4.3 Preserving logos

Preserve TEX, LATEX and LATEX 2$_\varepsilon$ logos.

\TeX

```
\let\@@TeX\TeX
\def\TeX{\@latin{\@@TeX}}
```

\LaTeX

```
\let\@@LaTeX\LaTeX
\def\LaTeX{\@latin{\@@LaTeX}}
```

\LaTeXe

```
\let\@@LaTeXe\LaTeXe
\def\LaTeXe{\@latin{\@@LaTeXe}}
```

### 60.4.4 List environments

List environments in Right-to-Left languages, are ticked and indented from the right instead of from the left. All the definitions that caused indentation are revised for Right-to-Left languages. LATEX keeps track on the indentation with the \leftmargin and \rightmargin values.

list Thus we need to override the definition of the \list macro: when in RTL mode, the right margins are the begining of the line.

```
\def\list#1#2{%
  \ifnum \@listdepth >5\relax
    \@toodeep
  \else
    \global\advance\@listdepth\@ne
  \fi
  \rightmargin\z@
  \listparindent\z@
  \itemindent\z@
  \csname @list\romannumeral\the\@listdepth\endcsname
  \def\@itemlabel{#1}%
  \let\makelabel\@mklab
  \@nmbrlistfalse
  #2\relax
```

```
\@trivlist
\parskip\parsep
\parindent\listparindent
\advance\linewidth -\rightmargin
\advance\linewidth -\leftmargin
```

The only change in the macro is the `\if@rl` case:

```
\if@rl
  \advance\@totalleftmargin \rightmargin
\else
  \advance\@totalleftmargin \leftmargin
\fi
\parshape \@ne \@totalleftmargin \linewidth
\ignorespaces}
```

`\labelenumii` `\p@enumiii` The `\labelenumii` and `\p@enumiii` commands use *parentheses.* They are revised to work Right-to-Left with the help of `\@brackets` macro defined above.

```
\def\labelenumii{\@brackets(\theenumii)}
\def\p@enumiii{\p@enumii\@brackets(\theenumii)}
```

### 60.4.5 Tables of moving stuff

Tables of moving arguments: table of contents (`toc`), list of figures (`lof`) and list of tables (`lot`) are handles here. These three default LaTeX tables will be used now exclusively for Left to Right stuff.

Three additional Right-to-Left tables: RL table of contents (`cot`), RL list of figures (`fol`), and RL list of tables (`tol`) are added. These three tables will be used exclusively for Right to Left stuff.

`\@tableofcontents` `\@listoffigures` `\@listoftables` We define 3 new macros similar to the standard LaTeX tables, but with one parameter — table file extension. These macros will help us to define our additional tables below.

```
\@ifclassloaded{letter}{}{% other
\@ifclassloaded{slides}{}{% other
  \@ifclassloaded{article}{% article
    \newcommand\@tableofcontents[1]{%
      \section*{\contentsname\@mkboth%
        {\MakeUppercase\contentsname}%
        {\MakeUppercase\contentsname}}%
      \@starttoc{#1}}
    \newcommand\@listoffigures[1]{%
      \section*{\listfigurename\@mkboth%
        {\MakeUppercase\listfigurename}%
        {\MakeUppercase\listfigurename}}%
      \@starttoc{#1}}
    \newcommand\@listoftables[1]{%
      \section*{\listtablename\@mkboth%
        {\MakeUppercase\listtablename}%
        {\MakeUppercase\listtablename}}%
    \@starttoc{#1}}}%
{% else report or book
  \newcommand\@tableofcontents[1]{%
    \@restonecolfalse\if@twocolumn\@restonecoltrue\onecolumn%
    \fi\chapter*{\contentsname\@mkboth%
      {\MakeUppercase\contentsname}%
      {\MakeUppercase\contentsname}}%
    \@starttoc{#1}\if@restonecol\twocolumn\fi}
  \newcommand\@listoffigures[1]{%
    \@restonecolfalse\if@twocolumn\@restonecoltrue\onecolumn%
    \fi\chapter*{\listfigurename\@mkboth%
      {\MakeUppercase\listfigurename}%
      {\MakeUppercase\listfigurename}}%
    \@starttoc{#1}\if@restonecol\twocolumn\fi}
  \newcommand\@listoftables[1]{%
    \if@twocolumn\@restonecoltrue\onecolumn\else\@restonecolfalse\fi%
    \chapter*{\listtablename\@mkboth%
      {\MakeUppercase\listtablename}%
      {\MakeUppercase\listtablename}}%
    \@starttoc{#1}\if@restonecol\twocolumn\fi}}%
```

`\lrtableofcontents` `\lrlistoffigures` `\lrlistoftables` Left-to-Right tables are called now `\lr`*xxx* and defined with the aid of three macros defined above (extensions `toc`, `lof`, and `lot`).

```
\newcommand\lrtableofcontents{\@tableofcontents{toc}}%
\newcommand\lrlistoffigures{\@listoffigures{lof}}%
\newcommand\lrlistoftables{\@listoftables{lot}}%
```

`\rltableofcontents` `\rllistoffigures` `\rllistoftables` Right-to-Left tables will be called `\rl`*xxx* and defined with the aid of three macros defined above (extensions `cot`, `fol`, and `tol`).

```
\newcommand\rltableofcontents{\@tableofcontents{cot}}%
\newcommand\rllistoffigures{\@listoffigures{fol}}%
\newcommand\rllistoftables{\@listoftables{tol}}%
```

`\tableofcontents` `\listoffigures` `\listoftables` Let `\`*xxx* be `\rl`*xxx* if the current direction is Right-to-Left and `\lr`*xxx* if it is Left-to-Right.

```
\renewcommand\tableofcontents{\if@rl\rltableofcontents%
                               \else\lrtableofcontents\fi}
\renewcommand\listoffigures{\if@rl\rllistoffigures%
                             \else\lrlistoffigures\fi}
\renewcommand\listoftables{\if@rl\rllistoftables%
                            \else\lrlistoftables\fi}}}
```

`\@dottedtocline` The following makes problems when making a Right-to-Left tables, since it uses `\leftskip` and `\rightskip` which are both mode dependent.

```
\def\@dottedtocline#1#2#3#4#5{%
  \ifnum #1>\c@tocdepth \else
    \vskip \z@ \@plus.2\p@
    {\if@rl\rightskip\else\leftskip\fi #2\relax
      \if@rl\leftskip\else\rightskip\fi \@tocrmarg \parfillskip
      -\if@rl\leftskip\else\rightskip\fi
     \parindent #2\relax\@afterindenttrue
     \interlinepenalty\@M
     \leavevmode
     \@tempdima #3\relax
     \advance\if@rl\rightskip\else\leftskip\fi \@tempdima
     \null\nobreak\hskip -\if@rl\rightskip\else\leftskip\fi
     {#4}\nobreak
     \leaders\hbox{$\m@th
        \mkern \@dotsep mu\hbox{.}\mkern \@dotsep
        mu$}\hfill
     \nobreak
     \hb@xt@\@pnumwidth{\hfil\normalfont \normalcolor \beginL#5\endL}%
     \par}%
  \fi}
```

`\l@part` This standard macro was redefined for table of contents since it uses `\rightskip` which is mode dependent.

```
\@ifclassloaded{letter}{}{% other
\@ifclassloaded{slides}{}{% other
\renewcommand*\l@part[2]{%
  \ifnum \c@tocdepth >-2\relax
    \addpenalty{-\@highpenalty}%
    \addvspace{2.25em \@plus\p@}%
    \begingroup
      \setlength\@tempdima{3em}%
      \parindent \z@ \if@rl\leftskip\else\rightskip\fi \@pnumwidth
      \parfillskip -\@pnumwidth
      {\leavevmode
       \large \bfseries #1\hfil \hb@xt@\@pnumwidth{\hss#2}}\par
       \nobreak
         \global\@nobreaktrue
         \everypar{\global\@nobreakfalse\everypar{}}%
    \endgroup
  \fi}}}
```

`\@part` Part is redefined to support new Right-to-Left table of contents (`cot`) as well as the Left-to-Right one (`toc`).

```
\@ifclassloaded{article}{% article class
  \def\@part[#1]#2{%
    \ifnum \c@secnumdepth >\m@ne
      \refstepcounter{part}%
      \addcontentsline{toc}{part}{\thepart\hspace{1em}#1}%
      \addcontentsline{cot}{part}{\thepart\hspace{1em}#1}%
    \else
      \addcontentsline{toc}{part}{#1}%
      \addcontentsline{cot}{part}{#1}%
    \fi
    {\parindent \z@ \raggedright
```

```
     \interlinepenalty \@M
     \normalfont
     \ifnum \c@secnumdepth >\m@ne
       \Large\bfseries \partname~\thepart
       \par\nobreak
     \fi
     \huge \bfseries #2%
     \markboth{}{}\par}%
    \nobreak
    \vskip 3ex
    \@afterheading}%
}{% report and book classes
  \def\@part[#1]#2{%
    \ifnum \c@secnumdepth >-2\relax
      \refstepcounter{part}%
      \addcontentsline{toc}{part}{\thepart\hspace{1em}#1}%
      \addcontentsline{cot}{part}{\thepart\hspace{1em}#1}%
    \else
      \addcontentsline{toc}{part}{#1}%
      \addcontentsline{cot}{part}{#1}%
    \fi
    \markboth{}{}%
    {\centering
     \interlinepenalty \@M
     \normalfont
     \ifnum \c@secnumdepth >-2\relax
       \huge\bfseries \partname~\thepart
       \par
       \vskip 20\p@
     \fi
     \Huge \bfseries #2\par}%
     \@endpart}}
```

`\@sect` Section was redefined from the `latex.ltx` file. It is changed to support both Left-to-Right (`toc`) and Right-to-Left (`cot`) table of contents simultaneously.

```
\def\@sect#1#2#3#4#5#6[#7]#8{%
  \ifnum #2>\c@secnumdepth
    \let\@svsec\@empty
  \else
    \refstepcounter{#1}%
    \protected@edef\@svsec{\@seccntformat{#1}\relax}%
  \fi
  \@tempskipa #5\relax
  \ifdim \@tempskipa>\z@
    \begingroup
      #6{%
        \@hangfrom{\hskip #3\relax\@svsec}%
          \interlinepenalty \@M #8\@@par}%
    \endgroup
    \csname #1mark\endcsname{#7}%
    \addcontentsline{toc}{#1}{%
      \ifnum #2>\c@secnumdepth \else
        \protect\numberline{\csname the#1\endcsname}%
      \fi
      #7}%
    \addcontentsline{cot}{#1}{%
      \ifnum #2>\c@secnumdepth \else
        \protect\numberline{\csname the#1\endcsname}%
      \fi
      #7}%
  \else
    \def\@svsechd{%
      #6{\hskip #3\relax
      \@svsec #8}%
      \csname #1mark\endcsname{#7}%
      \addcontentsline{toc}{#1}{%
        \ifnum #2>\c@secnumdepth \else
          \protect\numberline{\csname the#1\endcsname}%
        \fi
        #7}%
      \addcontentsline{cot}{#1}{%
        \ifnum #2>\c@secnumdepth \else
          \protect\numberline{\csname the#1\endcsname}%
        \fi
        #7}}%
  \fi
  \@xsect{#5}}
```

`\@caption` Caption was redefined from the `latex.ltx` file. It is changed to support Left-to-Right list of figures and list of tables (`lof` and `lot`) as well as new Right-to-Left lists (`fol` and `tol`) simultaneously.

```
\long\def\@caption#1[#2]#3{%
  \par
  \addcontentsline{\csname ext@#1\endcsname}{#1}%
    {\protect\numberline{\csname the#1\endcsname}%
    {\ignorespaces #2}}%
  \def\@fignm{figure}
  \ifx#1\@fignm\addcontentsline{fol}{#1}%
     {\protect\numberline{\csname the#1\endcsname}%
     {\ignorespaces #2}}\fi%
  \def\@tblnm{table}
  \ifx#1\@tblnm\addcontentsline{tol}{#1}%
     {\protect\numberline{\csname the#1\endcsname}%
     {\ignorespaces #2}}\fi%
  \begingroup
    \@parboxrestore
    \if@minipage
      \@setminipage
    \fi
    \normalsize
    \@makecaption{\csname fnum@#1\endcsname}{\ignorespaces #3}\par
  \endgroup}
```

\l@chapter This standard macro was redefined for table of contents since it uses `\rightskip` which is mode dependent.

```
\@ifclassloaded{letter}{}{%
\@ifclassloaded{slides}{}{%
  \@ifclassloaded{article}{}{%
    \renewcommand*\l@chapter[2]{%
      \ifnum \c@tocdepth >\m@ne
      \addpenalty{-\@highpenalty}%
      \vskip 1.0em \@plus\p@
      \setlength\@tempdima{1.5em}%
      \begingroup
         \parindent \z@ \if@rl\leftskip\else\rightskip\fi \@pnumwidth
         \parfillskip -\@pnumwidth
         \leavevmode \bfseries
         \advance\if@rl\rightskip\else\leftskip\fi\@tempdima
         \hskip -\if@rl\rightskip\else\leftskip\fi
         #1\nobreak\hfil \nobreak\hb@xt@\@pnumwidth{\hss#2}\par
         \penalty\@highpenalty
      \endgroup
      \fi}}}}
```

\l@section
\l@subsection
\l@subsubsection
\l@paragraph
\l@subparagraph

The toc entry for section did not work in article style. Also it does not print dots, which is funny when most of your work is divided into sections.

It was revised to use `\@dottedtocline` as in `report.sty` (by Yaniv Bargury) and was updated later for all kinds of sections (by Boris Lavva).

```
\@ifclassloaded{article}{%
\renewcommand*\l@section{\@dottedtocline{1}{1.5em}{2.3em}}
\renewcommand*\l@subsection{\@dottedtocline{2}{3.8em}{3.2em}}
\renewcommand*\l@subsubsection{\@dottedtocline{3}{7.0em}{4.1em}}
\renewcommand*\l@paragraph{\@dottedtocline{4}{10em}{5em}}
\renewcommand*\l@subparagraph{\@dottedtocline{5}{12em}{6em}}}{}
```

### 60.4.6 Two-column mode

This is the support of `twocolumn` option for the standard LaTeX 2$_\varepsilon$ classes. The following code was originally borrowed from the ArabTeX package, file `latexext.sty`, copyright by Klaus Lagally, Institut fuer Informatik, Universitaet Stuttgart. It was updated for this package by Boris Lavva.

\@outputdblcol
\set@outputdblcol
rl@outputdblcol

First column is `\@leftcolumn` will be shown at the right side, Second column is `\@outputbox` will be shown at the left side.

`\set@outputdblcol` IS CURRENTLY DISABLED. TODO: REMOVE IT [tzafrir]

```
\let\@@outputdblcol\@outputdblcol
%\def\set@outputdblcol{%
%  \if@rl\renewcommand{\@outputdblcol}{\rl@outputdblcol}%
%  \else\renewcommand{\@outputdblcol}{\@@outputdblcol}\fi}
\renewcommand{\@outputdblcol}{%
  \if@rlmain%
    \rl@outputdblcol%
  \else%
    \@@outputdblcol%
  \fi%
}
\newcommand{\rl@outputdblcol}{%
  \if@firstcolumn
    \global \@firstcolumnfalse
    \global \setbox\@leftcolumn \box\@outputbox
  \else
    \global \@firstcolumntrue
    \setbox\@outputbox \vbox {\hb@xt@\textwidth {%
                              \hskip\columnwidth%
                              \hfil\vrule\@width\columnseprule\hfil
                              \hb@xt@\columnwidth {%
                               \box\@leftcolumn \hss}%
                              \hb@xt@\columnwidth {%
                                \hskip-\textwidth%
                                \box\@outputbox \hss}%
                              \hskip\columnsep%
                              \hskip\columnwidth}}%
    \@combinedblfloats
    \@outputpage
    \begingroup
      \@dblfloatplacement
      \@startdblcolumn
      \@whilesw\if@fcolmade \fi
        {\@outputpage
         \@startdblcolumn}%
    \endgroup
 \fi}
```

### 60.4.7 Footnotes

`\footnoterule` The Right-to-Left footnote rule is simply reversed default Left-to-Right one. Footnotes can be used in RL or LR main modes, but changing mode while a footnote is pending is still unsolved.

```
\let\@@footnoterule=\footnoterule
\def\footnoterule{\if@rl\hb@xt@\hsize{\hss\vbox{\@@footnoterule}}%
                 \else\@@footnoterule\fi}
```

### 60.4.8 Headings and two-side support

When using `headings` or `myheadings` modes, we have to ensure that the language and direction of heading is the same as the whole chapter/part of the document.

This is implementing by setting special variable `\headlanguage` when starting new chapter/part.

In addition, when selecting the `twoside` option (default in `book` document class), the LR and RL modes need to be set properly for things on the heading and footing. This is done here too.

`ps@headings` `ps@myheadings` `headeven` `headodd`

First, we will support the standard `letter` class:

```
\@ifclassloaded{letter}{%
  \def\headodd{\protect\if@rl\beginR\fi\headtoname{}
              \ignorespaces\toname
              \hfil \@date
              \hfil \pagename{} \thepage\protect\if@rl\endR\fi}
  \if@twoside
     \def\ps@headings{%
         \let\@oddfoot\@empty\let\@evenfoot\@empty
         \def\@oddhead{\select@language{\headlanguage}\headodd}
         \let\@evenhead\@oddhead}
  \else
     \def\ps@headings{%
         \let\@oddfoot\@empty
         \def\@oddhead{\select@language{\headlanguage}\headodd}}
  \fi
  \def\headfirst{\protect\if@rl\beginR\fi\fromlocation \hfill %
                \telephonenum\protect\if@rl\endR\fi}
  \def\ps@firstpage{%
     \let\@oddhead\@empty
     \def\@oddfoot{\raisebox{-45\p@}[\z@]{%
        \hb@xt@\textwidth{\hspace*{100\p@}%
          \ifcase \@ptsize\relax
             \normalsize
          \or
             \small
          \or
             \footnotesize
          \fi
        \select@language{\headlanguage}\headfirst}}\hss}}
%
  \renewcommand{\opening}[1]{%
     \let\headlanguage=\languagename%
     \ifx\@empty\fromaddress%
        \thispagestyle{firstpage}%
        {\raggedleft\@date\par}%
     \else  % home address
        \thispagestyle{empty}%
        {\raggedleft
        \if@rl\begin{tabular}{@{\beginR\csname%
          to\@rllanguagename\endcsname}r@{\endR}}\ignorespaces
           \fromaddress \\*[2\parskip]%
           \@date \end{tabular}\par%
        \else\begin{tabular}{l}\ignorespaces
          \fromaddress \\*[2\parskip]%
           \@date \end{tabular}\par%
        \fi}%
     \fi
     \vspace{2\parskip}%
     {\raggedright \toname \\ \toaddress \par}%
     \vspace{2\parskip}%
     #1\par\nobreak}
}
```

Then, the `article`, `report` and `book` document classes are supported. Note, that in one-sided mode `\markright` was changed to `\markboth`.

```
{% article, report, book
  \def\headeven{\protect\if@rl\beginR\thepage\hfil\rightmark\endR
                \protect\else\thepage\hfil{\slshape\leftmark}
                \protect\fi}
  \def\headodd{\protect\if@rl\beginR\leftmark\hfil\thepage\endR
               \protect\else{\slshape\rightmark}\hfil\thepage
               \protect\fi}
  \@ifclassloaded{article}{% article
    \if@twoside   % two-sided
       \def\ps@headings{%
         \let\@oddfoot\@empty\let\@evenfoot\@empty
         \def\@evenhead{\select@language{\headlanguage}\headeven}%
         \def\@oddhead{\select@language{\headlanguage}\headodd}%
         \let\@mkboth\markboth
         \def\sectionmark##1{%
           \markboth {\MakeUppercase{%
               \ifnum \c@secnumdepth >\z@
                  \thesection\quad
               \fi
               ##1}}{}}%
         \def\subsectionmark##1{%
           \markright{%
             \ifnum \c@secnumdepth >\@ne
                \thesubsection\quad
             \fi
        ##1}}}
    \else          % one-sided
       \def\ps@headings{%
         \let\@oddfoot\@empty
         \def\@oddhead{\headodd}%
         \let\@mkboth\markboth
         \def\sectionmark##1{%
           \markboth{\MakeUppercase{%
               \ifnum \c@secnumdepth >\m@ne
                  \thesection\quad
               \fi
               ##1}}{\MakeUppercase{%
               \ifnum \c@secnumdepth >\m@ne
                  \thesection\quad
               \fi
               ##1}}}}
    \fi
%
    \def\ps@myheadings{%
      \let\@oddfoot\@empty\let\@evenfoot\@empty
      \def\@evenhead{\select@language{\headlanguage}\headeven}%
      \def\@oddhead{\select@language{\headlanguage}\headodd}%
      \let\@mkboth\@gobbletwo
      \let\sectionmark\@gobble
      \let\subsectionmark\@gobble
  }}{% report and book
    \if@twoside  % two-sided
       \def\ps@headings{%
         \let\@oddfoot\@empty\let\@evenfoot\@empty
         \def\@evenhead{\select@language{\headlanguage}\headeven}
         \def\@oddhead{\select@language{\headlanguage}\headodd}
         \let\@mkboth\markboth
         \def\chaptermark##1{%
           \markboth{\MakeUppercase{%
               \ifnum \c@secnumdepth >\m@ne
                  \@chapapp\ \thechapter. \ %
               \fi
               ##1}}{}}%
         \def\sectionmark##1{%
           \markright {\MakeUppercase{%
               \ifnum \c@secnumdepth >\z@
                  \thesection. \ %
               \fi
               ##1}}}}
    \else  % one-sided
       \def\ps@headings{%
         \let\@oddfoot\@empty
         \def\@oddhead{\select@language{\headlanguage}\headodd}
         \let\@mkboth\markboth
         \def\chaptermark##1{%
           \markboth{\MakeUppercase{%
               \ifnum \c@secnumdepth >\m@ne
                  \@chapapp\ \thechapter. \ %
               \fi
               ##1}}{\MakeUppercase{%
               \ifnum \c@secnumdepth >\m@ne
                  \@chapapp\ \thechapter. \ %
               \fi
               ##1}}}}
    \fi
    \def\ps@myheadings{%
      \let\@oddfoot\@empty\let\@evenfoot\@empty
      \def\@evenhead{\select@language{\headlanguage}\headeven}%
      \def\@oddhead{\select@language{\headlanguage}\headodd}%
      \let\@mkboth\@gobbletwo
      \let\chaptermark\@gobble
      \let\sectionmark\@gobble
  }}}
```

### 60.4.9 Postscript Porblems

Any command that is implemented by PostScript directives, e.g commands from the ps-tricks package, needs to be fixed, because the PostScript directives are being interpeted after the document has been converted by TEXto visual Hebrew (DVI, PostScript and PDF have visual Hebrew).

For instance: Suppose you wrote in your document:

`\textcolor{cyan}{some ltr text}`

This would be interpeted by TEXto something like:

`[postscript:make color cyan]some LTR text[postscript:make color black]`

However, with the bidirectionality support we get:

`\textcolor{cyan}{\hebalef\hebbet}`

Translated to:

`[postscript:make color black]{bet}{alef}[postscript:make color cyan]`

While we want:

`[postscript:make color cyan]{bet}{alef}[postscript:make color black]`

The following code will probably work at least with code that stays in the same line:

`@textcolor`

```
\AtBeginDocument{%
  %I assume that \@textcolor is only defined by the package color
  \ifx\@textcolor\@undefined\else%
    % If that macro was defined before the beginning of the document,
    % that is: the package was loaded: redefine it with bidi support
    \def\@textcolor#1#2#3{%
      \if@rl%
        \beginL\protect\leavevmode{\color#1{#2}\beginR#3\endR}\endL%
      \else%
        \protect\leavevmode{\color#1{#2}#3}%
      \fi%
    }%
  \fi%
}
% \end{macrocode}
% \end{macro}
% \begin{macro}{\thetrueSlideCounter}
%    This macro probably needs to be overriden for when using |prosper|,
%    (waiting for feedback. Tzafrir)
%    \begin{macrocode}
\@ifclassloaded{prosper}{%
```

```
  \def\thetrueSlideCounter{\arabicnorl{trueSlideCounter}}
}{}
```

### 60.4.10 Miscellaneous internal LaTeX macros

\raggedright \raggedleft

`\raggedright` was changed from `latex.ltx` file to support Right-to-Left mode, because of the bug in its implementation.

```
\def\raggedright{%
  \let\\\@centercr
  \leftskip\z@skip\rightskip\@flushglue
  \parindent\z@\parfillskip\z@skip}
```

Swap meanings of `\raggedright` and `\raggedleft` in Right-to-Left mode.

```
\let\@@raggedleft=\raggedleft
\let\@@raggedright=\raggedright
\renewcommand\raggedleft{\if@rl\@@raggedright%
                          \else\@@raggedleft\fi}
\renewcommand\raggedright{\if@rl\@@raggedleft%
                           \else\@@raggedright\fi}
```

\author

`\author` is inserted with `tabular` environment, and will be used in restricted horizontal mode. Therefore we have to add explicit direction change command when in Right-to-Left mode.

```
\let\@@author=\author
\renewcommand{\author}[1]{\@@author{\if@rl\beginR #1\endR\else #1\fi}}
```

\MakeUppercase \MakeLowercase

There are no uppercase and lowercase letters in most Right-to-Left languages, therefore we should redefine `\MakeUppercase` and `\MakeLowercase` LaTeX $2_\varepsilon$ commands.

```
\let\@@MakeUppercase=\MakeUppercase
\def\MakeUppercase#1{\if@rl#1\else\@@MakeUppercase{#1}\fi}
\let\@@MakeLowercase=\MakeLowercase
\def\MakeLowercase#1{\if@rl#1\else\@@MakeLowercase{#1}\fi}
```

\underline

We should explicitly use `\L` and `\R` commands in `\underline`d text.

```
\let\@@@underline=\underline
\def\underline#1{\@@@underline{\if@rl\R{#1}\else #1\fi}}
```

`\undertext` was added for LaTeX2.09 compatibility mode.

```
\if@compatibility
   \let\undertext=\underline
\fi
```

\@xnthm \@opargbegintheorem

The following has been inserted to correct the appearance of the number in `\newtheorem` to reorder theorem number components. A similar correction in the definition of `\@opargbegintheorem` was added too.

```
\def\@xnthm#1#2[#3]{%
  \expandafter\@ifdefinable\csname #1\endcsname
  {\@definecounter{#1}\@addtoreset{#1}{#3}%
    \expandafter\xdef\csname the#1\endcsname{\noexpand\@number
      {\expandafter\noexpand\csname the#3\endcsname \@thmcountersep
        \@thmcounter{#1}}}%
    \global\@namedef{#1}{\@thm{#1}{#2}}%
    \global\@namedef{end#1}{\@endtheorem}}
%
\def\@opargbegintheorem#1#2#3{%
  \trivlist
      \item[\hskip \labelsep{\bfseries #1\ #2\
          \@brackets({#3})}]\itshape}
```

`\@chapter` `\@schapter` The following was added for pretty printing of the chapter numbers, for supporting Right-to-Left tables (`cot`, `fol`, and `tol`), to save `\headlanguage` for use in running headers, and to start two-column mode depending on chapter's main language.

```
\@ifclassloaded{article}{}{%
  % For pretty priniting
  \def\@@chapapp{Chapter}
  \def\@@thechapter{\@@arabic\c@chapter}
  \def\@chapter[#1]#2{%
    \let\headlanguage=\languagename%
    %\set@outputdblcol%
    \ifnum \c@secnumdepth >\m@ne
       \refstepcounter{chapter}%
       \typeout{\@@chapapp\space\@@thechapter.}%
       \addcontentsline{toc}{chapter}%
       {\protect\numberline{\thechapter}#1}
       \addcontentsline{cot}{chapter}%
       {\protect\numberline{\thechapter}#1}
    \else
       \addcontentsline{toc}{chapter}{#1}%
       \addcontentsline{cot}{chapter}{#1}%
    \fi
    \chaptermark{#1}
    \addtocontents{lof}{\protect\addvspace{10\p@}}%
    \addtocontents{fol}{\protect\addvspace{10\p@}}%
    \addtocontents{lot}{\protect\addvspace{10\p@}}%
    \addtocontents{tol}{\protect\addvspace{10\p@}}%
    \if@twocolumn
       \@topnewpage[\@makechapterhead{#2}]%
    \else
       \@makechapterhead{#2}%
       \@afterheading
    \fi}
  %
  \def\@schapter#1{%
    \let\headlanguage=\languagename%
    %\set@outputdblcol%
    \if@twocolumn
       \@topnewpage[\@makeschapterhead{#1}]%
    \else
      \@makeschapterhead{#1}%
      \@afterheading
    \fi}}
```

\appendix Changed mainly for pretty printing of appendix numbers, and to start two-column mode with the right language (if needed).

```
\@ifclassloaded{letter}{}{% other
\@ifclassloaded{slides}{}{% other
  \@ifclassloaded{article}{% article
    \renewcommand\appendix{\par
      \setcounter{section}{0}%
      \setcounter{subsection}{0}%
      \renewcommand\thesection{\@Alph\c@section}}
  }{% report and book
    \renewcommand\appendix{\par
      %\set@outputdblcol%
      \setcounter{chapter}{0}%
      \setcounter{section}{0}%
      \renewcommand\@chapapp{\appendixname}%
      % For pretty priniting
      \def\@@chapapp{Appendix}%
      \def\@@thechapter{\@@Alph\c@chapter}
      \renewcommand\thechapter{\@Alph\c@chapter}}}}}
```

### 60.4.11 Bibliography and citations

\@cite \@biblabel \@lbibitem Citations are produced by the macro \@cite{*LABEL*}{*NOTE*}. Both the citation label and the note is typeset in the current direction. We have to use \@brackets macro in \@cite and \@biblabel macros. In addition, when using *alpha* or similar bibliography style, the \@lbibitem is used and have to be update to support bot Right-to-Left and Left-to-Right citations.

```
\def\@cite#1#2{\@brackets[{#1\if@tempswa , #2\fi}]}
\def\@biblabel#1{\@brackets[{#1}]}
\def\@lbibitem[#1]#2{\item[\@biblabel{#1}\hfill]\if@filesw
      {\let\protect\noexpand
       \immediate
       \if@rl\write\@auxout{\string\bibcite{#2}{\R{#1}}}%
       \else\write\@auxout{\string\bibcite{#2}{\L{#1}}}\fi%
      }\fi\ignorespaces}
```

thebibliography Use \rightmargin instead of \leftmargin when in RL mode.

```
\@ifclassloaded{letter}{}{% other
\@ifclassloaded{slides}{}{% other
\@ifclassloaded{article}{%
  \renewenvironment{thebibliography}[1]
  {\section*{\refname\@mkboth%
      {\MakeUppercase\refname}%
      {\MakeUppercase\refname}}%
```

```
    \list{\@biblabel{\@arabic\c@enumiv}}%
    {\settowidth\labelwidth{\@biblabel{#1}}%
      \if@rl\leftmargin\else\rightmargin\fi\labelwidth
      \advance\if@rl\leftmargin\else\rightmargin\fi\labelsep
      \@openbib@code
      \usecounter{enumiv}%
      \let\p@enumiv\@empty
      \renewcommand\theenumiv{\@arabic\c@enumiv}}%
    \sloppy
    \clubpenalty4000
    \@clubpenalty \clubpenalty
    \widowpenalty4000%
    \sfcode`\.\@m}
  {\def\@noitemerr
    {\@latex@warning{Empty `thebibliography' environment}}%
     \endlist}}%
{\renewenvironment{thebibliography}[1]{%
    \chapter*{\bibname\@mkboth%
      {\MakeUppercase\bibname}%
      {\MakeUppercase\bibname}}%
    \list{\@biblabel{\@arabic\c@enumiv}}%
    {\settowidth\labelwidth{\@biblabel{#1}}%
      \if@rl\leftmargin\else\rightmargin\fi\labelwidth
      \advance\if@rl\leftmargin\else\rightmargin\fi\labelsep
      \@openbib@code
      \usecounter{enumiv}%
      \let\p@enumiv\@empty
      \renewcommand\theenumiv{\@arabic\c@enumiv}}%
    \sloppy
    \clubpenalty4000
    \@clubpenalty \clubpenalty
    \widowpenalty4000%
    \sfcode`\.\@m}
  {\def\@noitemerr
    {\@latex@warning{Empty `thebibliography' environment}}%
     \endlist}}}}
```

`\@verbatim` All kinds of verbs (`\verb`,`\verb*`,`verbatim` and `verbatim*`) now can be used in Right-to-Left mode. Errors in latin mode solved too.

```
\def\@verbatim{%
  \let\do\@makeother \dospecials%
  \obeylines \verbatim@font \@noligs}
```

`\@makecaption` Captions are set always centered. This allows us to use bilingual captions, for example: `\caption{\R{RLtext} \\ \L{LRtext}}`, which will be formatted as:

Right to left caption here (RLtext)
Left to right caption here (LRtext)

See also `\bcaption` command below.

```
\long\def\@makecaption#1#2{%
  \vskip\abovecaptionskip%
  \begin{center}%
    #1: #2%
  \end{center} \par%
  \vskip\belowcaptionskip}
```

### 60.4.12 Additional bidirectional commands

- Section headings are typeset with the default global direction.
- Text in section headings in the reverse language *do not* have to be protected for the reflection command, as in: `\protect\L{`*Latin Text*`}`, because `\L` and `\R` are robust now.
- Table of contents, list of figures and list of tables should be typeset with the `\tableofcontents`, `\listoffigures` and `\listoftables` commands respectively.
- The above tables will be typeset in the main direction (and language) in effect where the above commands are placed.
- Only 2 tables of each kind are supported: one for Right-to-Left and another for Left-to-Right directions.

How to include line to both tables? One has to use bidirectional sectioning commands as following:

1. Use the `\b`*xxx* version of the sectioning commands in the text instead of the `\`*xxx* version (*xxx* is one of: `part`, `chapter`, `section`, `subsection`, `subsubsection`, `caption`).
2. Syntax of the `\b`*xxx* command is `\b`*xxx*`{`*RL text*`}{`*LR text*`}`. Both arguments are typeset in proper direction by default (no need to change direction for the text inside).
3. The section header inside the document will be typeset in the global direction in effect at the time. i.e. The `{`*RL text*`}` will be typeset if Right-to-Left mode is in effect and `{`*LR text*`}` otherwise.

`\bpart`

```
\newcommand{\bpart}[2]{\part{\protect\if@rl%
    #1 \protect\else #2 \protect\fi}}
```

`\bchapter`

```
\newcommand{\bchapter}[2]{\chapter{\protect\if@rl%
    #1 \protect\else #2 \protect\fi}}
```

`\bsection`

```
\newcommand{\bsection}[2]{\section{\protect\if@rl%
    #1 \protect\else #2 \protect\fi}}
```

\bsubsection

```
\newcommand{\bsubsection}[2]{\subsection{\protect\if@rl%
    #1 \protect\else #2 \protect\fi}}
```

\bsubsubsection

```
\newcommand{\bsubsubsection}[2]{\subsubsection{\protect\if@rl%
    #1 \protect\else #2 \protect\fi}}
```

\bcaption

```
\newcommand{\bcaption}[2]{%
  \caption[\protect\if@rl \R{#1}\protect\else \L{#2}\protect\fi]{%
    \if@rl\R{#1}\protect\\ \L{#2}
    \else\L{#2}\protect\\ \R{#1}\fi}}
```

The following definition is a modified version of `\bchapter`, meant as a bilingual twin for `\chapter*` and `\section*` (added by Irina Abramovici).

\bchapternn

```
\newcommand{\bchapternn}[2]{\chapter*{\protect\if@rl%
    #1 \protect\else #2 \protect\fi}}
```

\bsectionnn

```
\newcommand{\bsectionnn}[2]{\section*{\protect\if@rl%
    #1 \protect\else #2 \protect\fi}}
```

Finally, at end of babel package, the `\headlanguage` and two-column mode will be initialized according to the current language.

```
\AtEndOfPackage{\rlAtEndOfPackage}
%
\def\rlAtEndOfPackage{%
  \global\let\headlanguage=\languagename%\set@outputdblcol%
}
⟨/rightleft⟩
```

## 60.5 Hebrew calendar

The original version of the package `hebcal.sty`[70] for TeX and LaTeX2.09, entitled "TeX & LaTeX macros for computing Hebrew date from Gregorian one" was created by Michail Rozman, `misha@iop.tartu.ew.su`[71]

| | | |
|---|---|---|
| Released: | Tammuz 12, 5751–June 24, 1991 | |
| Corrected: | Shebat 10, 5752–January 15, 1992 | by Rama Porrat |
| Corrected: | Adar II 5, 5752–March 10, 1992 | by Misha |
| Corrected: | Tebeth, 5756–January 1996 | Dan Haran (haran@math.tau.ac.il) |

[70]The following description of `hebcal` package is based on the comments included with original source by the author, Michail Rozman.

[71]Please direct any comments, bug reports, questions, etc. about the package to this address.

The package was adjusted for babel and LaTeX 2ε by Boris Lavva.

Changes to the printing routine (only) by Ron Artstein, June 1, 2003.

This package should be included *after* the `babel` with `hebrew` option, as following:

```
\documentclass[...]{...}
\usepackage[hebrew,...,other languages, ...]{babel}
\usepackage{hebcal}
```

Two main user-level commands are provided by this package:

`\Hebrewtoday` Computes today's Hebrew date and prints it. If we are presently in Hebrew mode, the date will be printed in Hebrew, otherwise — in English (like Shebat 10, 5752).

`\Hebrewdate` Computes the Hebrew date from the given Gregorian date and prints it. If we are presently in Hebrew mode, the date will be printed in Hebrew, otherwise — in English (like Shebat 10, 5752). An example of usage is shown below:

```
\newcount\hd \newcount\hm \newcount\hy
\hd=10 \hm=3 \hy=1992
\Hebrewdate{\hd}{\hm}{\hy}
```

`full` The package option `full` sets the flag `\@full@hebrew@year`, which causes years from the current millenium to be printed with the thousands digit (he-tav-shin-samekh-gimel). Without this option, thousands are not printed for the current millenium. NOTE: should this be a command option rather than a package option? –RA.

### 60.5.1 Introduction

The Hebrew calendar is inherently complicated: it is lunisolar – each year starts close to the autumn equinox, but each month must strictly start at a new moon. Thus Hebrew calendar must be harmonized simultaneously with both lunar and solar events. In addition, for reasons of the religious practice, the year cannot start on Sunday, Wednesday or Friday.

For the full description of Hebrew calendar and for the list of references see:

> Nachum Dershowitz and Edward M. Reingold, *"Calendarical Calculations"*, Software–Pract.Exper., vol. 20 (9), pp.899–928 (September 1990).

`C` translation of `LISP` programs from the above article available from Mr. Wayne Geiser, `geiser%pictel@uunet.uu.net`.

The 4th distribution (July 1989) of hdate/hcal (Hebrew calendar programs similar to UNIX date/cal) by Mr. Amos Shapir, `amos@shum.huji.ac.il`, contains short and very clear description of algorithms.

### 60.5.2 Registers, Commands, Formatting Macros

The command `\Hebrewtoday` produces today's date for Hebrew calendar. It is similar to the standard LaTeX 2$_\varepsilon$ command `\today`. In addition three numerical registers `\Hebrewday`, `\Hebrewmonth` and `\Hebrewyear` are set. For setting this registers without producing of date string command `\Hebrewsetreg` can be used.

The command `\Hebrewdate{`*Gday*`}{`*Gmonth*`}{`*Gyear*`}` produces Hebrew calendar date corresponding to Gregorian date `Gday.Gmonth.Gyear`. Three numerical registers `\Hebrewday`, `\Hebrewmonth` and `\Hebrewyear` are set.

For converting arbitrary Gregorian date `Gday.Gmonth.Gyear` to Hebrew date `Hday.Hmonth.Hyear` without producing date string the command:

`\HebrewFromGregorian{`*Gday*`}{`*Gmonth*`}{ `*Gyear*`}{`*Hday*`}{`*Hmonth*`}{`*Hyear*`}`

can be used.

```
⟨*calendar⟩
\newif\if@full@hebrew@year
\@full@hebrew@yearfalse
\DeclareOption{full}{\@full@hebrew@yeartrue}
\ProcessOptions
\newcount\Hebrewday  \newcount\Hebrewmonth   \newcount\Hebrewyear
```

`\Hebrewdate` Hebrew calendar date corresponding to Gregorian date `Gday.Gmonth.Gyear`. If Hebrew (right-to-left) fonts & macros are not loaded, we have to use English format.

```
\def\Hebrewdate#1#2#3{%
    \HebrewFromGregorian{#1}{#2}{#3}
                        {\Hebrewday}{\Hebrewmonth}{\Hebrewyear}%
    \ifundefined{if@rl}%
       \FormatForEnglish{\Hebrewday}{\Hebrewmonth}{\Hebrewyear}%
    \else%
       \FormatDate{\Hebrewday}{\Hebrewmonth}{\Hebrewyear}%
    \fi}
```

`\Hebrewtoday` Today's date in Hebrew calendar.

```
\def\Hebrewtoday{\Hebrewdate{\day}{\month}{\year}}
\let\hebrewtoday=\Hebrewtoday
```

`\Hebrewsetreg` Set registers: today's date in hebrew calendar.

```
\def\Hebrewsetreg{%
    \HebrewFromGregorian{\day}{\month}{\year}
                        {\Hebrewday}{\Hebrewmonth}{\Hebrewyear}}
```

`\FormatDate` Prints a Hebrew calendar date `Hebrewday.Hebrewmonth.Hebrewyear`.

```
\def\FormatDate#1#2#3{%
        \if@rl%
            \FormatForHebrew{#1}{#2}{#3}%
        \else%
            \FormatForEnglish{#1}{#2}{#3}
        \fi}
```

To prepare another language version of Hebrew calendar commands, one should change or add commands here.

We start with Hebrew language macros.

`\HebrewYearName` Prints Hebrew year as a Hebrew number. Disambiguates strings by adding lamed-pe-gimel to years of the first Jewish millenium and to years divisible by 1000. Suppresses the thousands digit in the current millenium unless the package option `full` is selected. NOTE: should this be provided as a command option rather than a package option? –RA.

```
\def\HebrewYearName#1{{%
  \@tempcnta=#1\divide\@tempcnta by 1000\multiply\@tempcnta by 1000
  \ifnum#1=\@tempcnta\relax % divisible by 1000: disambiguate
    \Hebrewnumeralfinal{#1}\ )\heblamed\hebpe"\hebgimel(%
  \else % not divisible by 1000
    \ifnum#1<1000\relax     % first millennium: disambiguate
      \Hebrewnumeralfinal{#1}\ )\heblamed\hebpe"\hebgimel(%
    \else
      \ifnum#1<5000
        \Hebrewnumeralfinal{#1}%
      \else
        \ifnum#1<6000 % current millenium, print without thousands
          \@tempcnta=#1\relax
          \if@full@hebrew@year\else\advance\@tempcnta by -5000\fi
          \Hebrewnumeralfinal{\@tempcnta}%
        \else % #1>6000
          \Hebrewnumeralfinal{#1}%
        \fi
      \fi
    \fi
  \fi}}
```

`\HebrewMonthName` The macro `\HebrewMonthName`{*month*}{*year*} returns the name of month in the 'year'.

```
\def\HebrewMonthName#1#2{%
   \ifnum #1 = 7 %
   \CheckLeapHebrewYear{#2}%
       \if@HebrewLeap \hebalef\hebdalet\hebresh\ \hebbet'%
          \else \hebalef\hebdalet\hebresh%
       \fi%
   \else%
       \ifcase#1%
          % nothing for 0
          \or\hebtav\hebshin\hebresh\hebyod%
          \or\hebhet\hebshin\hebvav\hebfinalnun%
          \or\hebkaf\hebsamekh\heblamed\hebvav%
          \or\hebtet\hebbet\hebtav%
          \or\hebshin\hebbet\hebtet%
          \or\hebalef\hebdalet\hebresh\ \hebalef'%
          \or\hebalef\hebdalet\hebresh\ \hebbet'%
```

```
            \or\hebnun\hebyod\hebsamekh\hebfinalnun%
            \or\hebalef\hebyod\hebyod\hebresh%
            \or\hebsamekh\hebyod\hebvav\hebfinalnun%
            \or\hebtav\hebmem\hebvav\hebzayin%
            \or\hebalef\hebbet%
            \or\hebalef\heblamed\hebvav\heblamed%
        \fi%
    \fi}
```

`\HebrewDayName` Name of day in Hebrew letters (gimatria).

```
\def\HebrewDayName#1{\Hebrewnumeral{#1}}
```

`\FormatForHebrew` The macro `\FormatForHebrew{`*hday*`}{`*hmonth* `}{`*hyear*`}` returns the formatted Hebrew date in Hebrew language.

```
\def\FormatForHebrew#1#2#3{%
  \HebrewDayName{#1}~\hebbet\HebrewMonthName{#2}{#3},~%
  \HebrewYearName{#3}}
```

We continue with two English language macros for Hebrew calendar.

`\HebrewMonthNameInEnglish` The macro `\HebrewMonthNameInEnglish{`*month*`}{` *year*`}` is similar to `\HebrewMonthName` described above. It returns the name of month in the Hebrew 'year' in English.

```
\def\HebrewMonthNameInEnglish#1#2{%
    \ifnum #1 = 7%
    \CheckLeapHebrewYear{#2}%
        \if@HebrewLeap Adar II\else Adar\fi%
    \else%
        \ifcase #1%
            % nothing for 0
            \or Tishrei%
            \or Heshvan%
            \or Kislev%
            \or Tebeth%
            \or Shebat%
            \or Adar I%
            \or Adar II%
            \or Nisan%
            \or Iyar%
            \or Sivan%
            \or Tammuz%
            \or Av%
            \or Elul%
        \fi
    \fi}
```

`\FormatForEnglish` The macro `\FormatForEnglish{`*hday*`}{`*hmonth* `}{`*hyear*`}` is similar to `\FormatForHebrew` macro described above and returns the formatted Hebrew date in English.

```
\def\FormatForEnglish#1#2#3{%
    \HebrewMonthNameInEnglish{#2}{#3} \number#1,\ \number#3}
```

### 60.5.3 Auxiliary Macros

```
\newcount\@common
```

`\Remainder` `\Remainder{`$a$`}{`$b$`}{`$c$`}` calculates $c = a\%b == a - b \times \frac{a}{b}$

```
\def\Remainder#1#2#3{%
    #3 = #1%                  %  c = a
    \divide #3 by #2%         %  c = a/b
    \multiply #3 by -#2%      %  c = -b(a/b)
    \advance #3 by #1}%       %  c = a - b(a/b)
```

```
\newif\if@Divisible
```

`\CheckIfDivisible` `\CheckIfDivisible{`$a$`}{`$b$`}` sets `\@Divisibletrue` if $a\%b == 0$

```
\def\CheckIfDivisible#1#2{%
    {%
      \countdef\tmp = 0% \tmp == \count0 - temporary variable
      \Remainder{#1}{#2}{\tmp}%
      \ifnum \tmp = 0%
          \global\@Divisibletrue%
      \else%
          \global\@Divisiblefalse%
      \fi}}
```

`\ifundefined` From the TEXbook, ex. 7.7:

`\ifundefined{`*command*`}<true text>\else<false text>\fi`

```
\def\ifundefined#1{\expandafter\ifx\csname#1\endcsname\relax}
```

### 60.5.4 Gregorian Part

```
\newif\if@GregorianLeap
```

`\IfGregorianLeap` Conditional which is true if Gregorian 'year' is a leap year: $((year\%4 == 0) \wedge (year\%100 \neq 0)) \vee (year\%400 == 0)$

```
\def\IfGregorianLeap#1{%
    \CheckIfDivisible{#1}{4}%
    \if@Divisible%
        \CheckIfDivisible{#1}{100}%
        \if@Divisible%
            \CheckIfDivisible{#1}{400}%
            \if@Divisible%
                \@GregorianLeaptrue%
            \else%
                \@GregorianLeapfalse%
            \fi%
        \else%
```

```
            \@GregorianLeaptrue%
        \fi%
    \else%
        \@GregorianLeapfalse%
    \fi%
    \if@GregorianLeap}
```

\GregorianDaysInPriorMonths The macro `\GregorianDaysInPriorMonths{`*month*`}{`*year*`}{`*days*`}` calculates the number of days in months prior to 'month' in the 'year'.

```
\def\GregorianDaysInPriorMonths#1#2#3{%
    {%
        #3 = \ifcase #1%
               0 \or%             % no month number 0
               0 \or%
              31 \or%
              59 \or%
              90 \or%
             120 \or%
             151 \or%
             181 \or%
             212 \or%
             243 \or%
             273 \or%
             304 \or%
             334%
        \fi%
        \IfGregorianLeap{#2}%
            \ifnum #1 > 2%        % if month after February
                \advance #3 by 1% % add leap day
            \fi%
        \fi%
        \global\@common = #3}%
    #3 = \@common}
```

\GregorianDaysInPriorYears The macro `\GregorianDaysInPriorYears{`*year*`}{`*days*`}` calculates the number of days in years prior to the 'year'.

```
\def\GregorianDaysInPriorYears#1#2{%
     {%
         \countdef\tmpc = 4%      % \tmpc==\count4
         \countdef\tmpb = 2%      % \tmpb==\count2
         \tmpb = #1%              %
         \advance \tmpb by -1%    %
         \tmpc = \tmpb%           % \tmpc = \tmpb = year-1
         \multiply \tmpc by 365%  % Days in prior years =
         #2 = \tmpc%              % = 365*(year-1) ...
         \tmpc = \tmpb%           %
         \divide \tmpc by 4%      % \tmpc = (year-1)/4
         \advance #2 by \tmpc%    % ... plus Julian leap days ...
         \tmpc = \tmpb%           %
```

```
        \divide \tmpc by 100%    % \tmpc = (year-1)/100
        \advance #2 by -\tmpc%   % ... minus century years ...
        \tmpc = \tmpb%           %
        \divide \tmpc by 400%    % \tmpc = (year-1)/400
        \advance #2 by \tmpc%    % ... plus 4-century years.
        \global\@common = #2}%
    #2 = \@common}
```

\AbsoluteFromGregorian The macro \AbsoluteFromGregorian{*day*}{*month*}{*year*}{*absdate*} calculates the absolute date (days since 01.01.0001) from Gregorian date `day.month.year`.

```
\def\AbsoluteFromGregorian#1#2#3#4{%
    {%
        \countdef\tmpd = 0%        % \tmpd==\count0
        #4 = #1%                   % days so far this month
        \GregorianDaysInPriorMonths{#2}{#3}{\tmpd}%
        \advance #4 by \tmpd%      % add days in prior months
        \GregorianDaysInPriorYears{#3}{\tmpd}%
        \advance #4 by \tmpd%      % add days in prior years
        \global\@common = #4}%
    #4 = \@common}
```

### 60.5.5 Hebrew Part

```
\newif\if@HebrewLeap
```

\CheckLeapHebrewYear Set `\@HebrewLeaptrue` if Hebrew 'year' is a leap year, i.e. if $(1+7\times \mathit{year})\%19 < 7$ then *true* else *false*

```
\def\CheckLeapHebrewYear#1{%
    {%
        \countdef\tmpa = 0%        % \tmpa==\count0
        \countdef\tmpb = 1%        % \tmpb==\count1
%
        \tmpa = #1%
        \multiply \tmpa by 7%
        \advance \tmpa by 1%
        \Remainder{\tmpa}{19}{\tmpb}%
        \ifnum \tmpb < 7%          % \tmpb = (7*year+1)%19
            \global\@HebrewLeaptrue%
        \else%
            \global\@HebrewLeapfalse%
        \fi}}
```

\HebrewElapsedMonths The macro \HebrewElapsedMonths{*year*}{*months*} determines the number of months elapsed from the Sunday prior to the start of the Hebrew calendar to the mean conjunction of Tishri of Hebrew 'year'.

```
\def\HebrewElapsedMonths#1#2{%
    {%
        \countdef\tmpa = 0%        % \tmpa==\count0
        \countdef\tmpb = 1%        % \tmpb==\count1
        \countdef\tmpc = 2%      % \tmpc==\count2
%
        \tmpa = #1%              %
        \advance \tmpa by -1%    %
        #2 = \tmpa%              % #2 = \tmpa = year-1
        \divide #2 by 19%        % Number of complete Meton cycles
        \multiply #2 by 235%     % #2 = 235*((year-1)/19)
%
        \Remainder{\tmpa}{19}{\tmpb}% \tmpa = years%19-years this cycle
        \tmpc = \tmpb%           %
        \multiply \tmpb by 12%   %
        \advance #2 by \tmpb%    % add regular months this cycle
%
        \multiply \tmpc by 7%    %
        \advance \tmpc by 1%     %
        \divide \tmpc by 19%     % \tmpc = (1+7*((year-1)%19))/19 -
%                                %  number of leap months this cycle
        \advance #2 by \tmpc%    %  add leap months
%
        \global\@common = #2}%
    #2 = \@common}
```

\HebrewElapsedDays The macro \HebrewElapsedDays{*year*}{*days*} determines the number of days elapsed from the Sunday prior to the start of the Hebrew calendar to the mean conjunction of Tishri of Hebrew 'year'.

```
\def\HebrewElapsedDays#1#2{%
    {%
        \countdef\tmpa = 0%      % \tmpa==\count0
        \countdef\tmpb = 1%      % \tmpb==\count1
        \countdef\tmpc = 2%      % \tmpc==\count2
%
        \HebrewElapsedMonths{#1}{#2}%
        \tmpa = #2%              %
        \multiply \tmpa by 13753% %
        \advance \tmpa by 5604%  % \tmpa=MonthsElapsed*13758 + 5604
        \Remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
        \divide \tmpa by 25920%
%
        \multiply #2 by 29%
        \advance #2 by 1%
        \advance #2 by \tmpa%    %  #2 = 1 + MonthsElapsed*29 +
%                                %          PartsElapsed/25920
        \Remainder{#2}{7}{\tmpa}% %  \tmpa == DayOfWeek
        \ifnum \tmpc < 19440%
            \ifnum \tmpc < 9924%
            \else%               % New moon at 9 h. 204 p. or later
                \ifnum \tmpa = 2% % on Tuesday ...
                    \CheckLeapHebrewYear{#1}% of a common year
                    \if@HebrewLeap%
                    \else%
```

```
                        \advance #2 by 1%
                    \fi%
                \fi%
            \fi%
            \ifnum \tmpc < 16789%
            \else%                 % New moon at 15 h. 589 p. or later
                \ifnum \tmpa = 1%  % on Monday ...
                    \advance #1 by -1%
                    \CheckLeapHebrewYear{#1}% at the end of leap year
                    \if@HebrewLeap%
                        \advance #2 by 1%
                    \fi%
                \fi%
            \fi%
        \else%
            \advance #2 by 1%      %  new moon at or after midday
        \fi%
%
        \Remainder{#2}{7}{\tmpa}%  %  \tmpa == DayOfWeek
        \ifnum \tmpa = 0%          %  if Sunday ...
            \advance #2 by 1%
        \else%                     %
            \ifnum \tmpa = 3%      %  Wednesday ...
                \advance #2 by 1%
            \else%
                \ifnum \tmpa = 5%  %  or Friday
                     \advance #2 by 1%
                \fi%
            \fi%
        \fi%
        \global\@common = #2}%
    #2 = \@common}
```

`\DaysInHebrewYear` The macro `\DaysInHebrewYear{`*year*`}{`*days*`}` calculates the number of days in Hebrew 'year'.

```
\def\DaysInHebrewYear#1#2{%
    {%
        \countdef\tmpe = 12%   % \tmpe==\count12
%
        \HebrewElapsedDays{#1}{\tmpe}%
        \advance #1 by 1%
        \HebrewElapsedDays{#1}{#2}%
        \advance #2 by -\tmpe%
        \global\@common = #2}%
    #2 = \@common}
```

`\HebrewDaysInPriorMonths` The macro `\HebrewDaysInPriorMonths{`*month*`}{`*year*`}{`*days*`}` calculates the number of days in months prior to 'month' in the 'year'.

```
\def\HebrewDaysInPriorMonths#1#2#3{%
```

```
    {%
        \countdef\tmpf= 14%    % \tmpf==\count14
%
        #3 = \ifcase #1%       % Days in prior month of regular year
               0 \or%          % no month number 0
               0 \or%          % Tishri
              30 \or%          % Heshvan
              59 \or%          % Kislev
              89 \or%          % Tebeth
             118 \or%          % Shebat
             148 \or%          % Adar I
             148 \or%          % Adar II
             177 \or%          % Nisan
             207 \or%          % Iyar
             236 \or%          % Sivan
             266 \or%          % Tammuz
             295 \or%          % Av
             325 \or%          % Elul
             400%              % Dummy
        \fi%
        \CheckLeapHebrewYear{#2}%
        \if@HebrewLeap%            % in leap year
            \ifnum #1 > 6%         % if month after Adar I
                \advance #3 by 30% % add  30 days
            \fi%
        \fi%
        \DaysInHebrewYear{#2}{\tmpf}%
        \ifnum #1 > 3%
            \ifnum \tmpf = 353%    %
                \advance #3 by -1% %
            \fi%                   %  Short Kislev
            \ifnum \tmpf = 383%    %
                \advance #3 by -1% %
            \fi%                   %
        \fi%
%
        \ifnum #1 > 2%
            \ifnum \tmpf = 355%    %
                \advance #3 by 1%  %
            \fi%                   %  Long Heshvan
            \ifnum \tmpf = 385%    %
                \advance #3 by 1%  %
            \fi%                   %
        \fi%
        \global\@common = #3}%
    #3 = \@common}
```

`\AbsoluteFromHebrew` The macro `\AbsoluteFromHebrew{`*day*`}{`*month*`}{`*year*`}{`*absdate*`}` calculates the absolute date of Hebrew date `day.month.year`.

```
\def\AbsoluteFromHebrew#1#2#3#4{%
```

```
    {%
        #4 = #1%
        \HebrewDaysInPriorMonths{#2}{#3}{#1}%
        \advance #4 by #1%         % Add days in prior months this year
        \HebrewElapsedDays{#3}{#1}%
        \advance #4 by #1%         % Add days in prior years
        \advance #4 by -1373429%   % Subtract days before Gregorian
        \global\@common = #4}%     %   01.01.0001
    #4 = \@common}
```

\HebrewFromGregorian The macro \HebrewFromGregorian{*Gday*}{*Gmonth*}{*Gyear*}{*Hday*}{*Hmonth*}-{*Hyear*} evaluates Hebrew date Hday, Hmonth, Hyear from Gregorian date Gday, Gmonth, Gyear.

```
\def\HebrewFromGregorian#1#2#3#4#5#6{%
    {%
        \countdef\tmpx= 17%        % \tmpx==\count17
        \countdef\tmpy= 18%        % \tmpy==\count18
        \countdef\tmpz= 19%        % \tmpz==\count19
%
        #6 = #3%                   %
        \global\advance #6 by 3761%  approximation from above
        \AbsoluteFromGregorian{#1}{#2}{#3}{#4}%
        \tmpz = 1  \tmpy = 1%
        \AbsoluteFromHebrew{\tmpz}{\tmpy}{#6}{\tmpx}%
        \ifnum \tmpx > #4%              %
            \global\advance #6 by -1% Hyear = Gyear + 3760
            \AbsoluteFromHebrew{\tmpz}{\tmpy}{#6}{\tmpx}%
        \fi%                            %
        \advance #4 by -\tmpx%     % Days in this year
        \advance #4 by 1%          %
        #5 = #4%                   %
        \divide #5 by 30%          % Approximation for month from below
        \loop%                     % Search for month
            \HebrewDaysInPriorMonths{#5}{#6}{\tmpx}%
            \ifnum \tmpx < #4%
                \advance #5 by 1%
                \tmpy = \tmpx%
        \repeat%
        \global\advance #5 by -1%
        \global\advance #4 by -\tmpy}}
⟨/calendar⟩
```

## 61 Hebrew input encodings

Hebrew input encodings defined in file hebinp.dtx[72] should be used with inputenc LaTeX 2$_\varepsilon$ package. This package allows the user to specify an input

[72]The files described in this section have version number v1.1b and were last revised on 2004/02/20.

encoding from this file (for example, ISO Hebrew/Latin 8859-8, IBM Hebrew codepage 862 or MS Windows Hebrew codepage 1255) by saying:

```
\usepackage[encoding name]{inputenc}
```

The encoding can also be selected in the document with:

```
\inputencoding{encoding name}
```

The only practical use of this command within a document is when using text from several documents to build up a composite work such as a volume of journal articles. Therefore this command will be used only in vertical mode.

The encodings provided by this package are:

- `si960` 7-bit Hebrew encoding for the range 32–127. This encoding also known as "old-code" and defined by Israeli Standard SI-960.
- `8859-8` ISO 8859-8 Hebrew/Latin encoding commonly used in UNIX systems. This encoding also known as "new-code" and includes hebrew letters in positions starting from 224.
- `cp862` IBM 862 code page commonly used by DOS on IBM-compatible personal computers. This encoding also known as "pc-code" and includes hebrew letters in positions starting from 128.
- `cp1255` MS Windows 1255 (hebrew) code page which is similar to 8859-8. In addition to hebrew letters, this encoding contains also hebrew vowels and dots (nikud).

Each encoding has an associated `.def` file, for example `8859-8.def` which defines the behaviour of each input character, using the commands:

```
\DeclareInputText{slot}{text}
\DeclareInputMath{slot}{math}
```

This defines the input character *slot* to be the *text* material or *math* material respectively. For example, `8859-8.def` defines slots `"EA` (letter hebalef) and `"B5` ($\mu$) by saying:

```
\DeclareInputText{224}{\hebalef}
\DeclareInputMath{181}{\mu}
```

Note that the *commands* should be robust, and should not be dependent on the output encoding. The same *slot* should not have both a text and a math declaration for it. (This restriction may be removed in future releases of inputenc).

The `.def` file may also define commands using the declarations: `\providecommand` or `\ProvideTextCommandDefault`. For example, `8859-8.def` defines:

```
\ProvideTextCommandDefault{\textonequarter}{\ensuremath{\frac14}}
\DeclareInputText{188}{\textonequarter}
```

The use of the 'provide' forms here will ensure that a better definition will not be over-written; their use is recommended since, in general, the best defintion depends on the fonts available.

See the documentation in `inputenc.dtx` for details of how to declare input definitions for various encodings.

## 61.1 Default definitions for characters

First, we insert a `\makeatletter` at the beginning of all `.def` files to use `@` symbol in the macros' names.

```
⟨−driver⟩\makeatletter
```

Some input characters map to internal functions which are not in either the `T1` or `OT1` font encoding. For this reason default definitions are provided in the encoding file: these will be used unless some other output encoding is used which supports those glyphs. In some cases this default defintion has to be simply an error message.

Note that this works reasonably well only because the encoding files for both `OT1` and `T1` are loaded in the standard LaTeX format.

```
⟨*8859 − 8 | cp862 | cp1255⟩
\ProvideTextCommandDefault{\textdegree}{\ensuremath{{^\circ}}}
\ProvideTextCommandDefault{\textonehalf}{\ensuremath{\frac12}}
\ProvideTextCommandDefault{\textonequarter}{\ensuremath{\frac14}}
⟨/8859 − 8 | cp862 | cp1255⟩
⟨*8859 − 8 | cp1255⟩
\ProvideTextCommandDefault{\textthreequarters}{\ensuremath{\frac34}}
⟨/8859 − 8 | cp1255⟩
⟨*cp862 | cp1255⟩
\ProvideTextCommandDefault{\textflorin}{\textit{f}}
⟨/cp862 | cp1255⟩
⟨*cp862⟩
\ProvideTextCommandDefault{\textpeseta}{Pt}
⟨/cp862⟩
```

The name `\textblacksquare` is derived from the AMS symbol name since Adobe seem not to want this symbol. The default definition, as a rule, makes no claim to being a good design.

```
⟨*cp862⟩
\ProvideTextCommandDefault{\textblacksquare}
   {\vrule \@width .3em \@height .4em \@depth -.1em\relax}
⟨/cp862⟩
```

Some commands can't be faked, so we have them generate an error message.

```
⟨*8859 − 8 | cp862 | cp1255⟩
\ProvideTextCommandDefault{\textcent}
   {\TextSymbolUnavailable\textcent}
\ProvideTextCommandDefault{\textyen}
   {\TextSymbolUnavailable\textyen}
⟨/8859 − 8 | cp862 | cp1255⟩
```

```
⟨*8859 − 8⟩
\ProvideTextCommandDefault{\textcurrency}
   {\TextSymbolUnavailable\textcurrency}
⟨/8859 − 8⟩
⟨*cp1255⟩
\ProvideTextCommandDefault{\newsheqel}
   {\TextSymbolUnavailable\newsheqel}
⟨/cp1255⟩
⟨*8859 − 8 | cp1255⟩
\ProvideTextCommandDefault{\textbrokenbar}
   {\TextSymbolUnavailable\textbrokenbar}
⟨/8859 − 8 | cp1255⟩
⟨*cp1255⟩
\ProvideTextCommandDefault{\textperthousand}
   {\TextSymbolUnavailable\textperthousand}
⟨/cp1255⟩
```

Characters that are supposed to be used only in math will be defined by `\providecommand` because LaTeX $2_\varepsilon$ assumes that the font encoding for math fonts is static.

```
⟨*8859 − 8 | cp1255⟩
\providecommand{\mathonesuperior}{{^1}}
\providecommand{\maththreesuperior}{{^3}}
⟨/8859 − 8 | cp1255⟩
⟨*8859 − 8 | cp862 | cp1255⟩
\providecommand{\mathtwosuperior}{{^2}}
⟨/8859 − 8 | cp862 | cp1255⟩
⟨*cp862⟩
\providecommand{\mathordmasculine}{{^o}}
\providecommand{\mathordfeminine}{{^a}}
⟨/cp862⟩
```

## 61.2 The SI-960 encoding

The SI-960 or "old-code" encoding only allows characters in the range 32–127, so we only need to provide an empty `si960.def` file.

## 61.3 The ISO 8859-8 encoding and the MS Windows cp1255 encoding

The `8859-8.def` encoding file defines the characters in the ISO 8859-8 encoding.

The MS Windows Hebrew character set incorporates the Hebrew letter repertoire of ISO 8859-8, and uses the same code points (starting from 224). It has also some important additions in the 128–159 and 190–224 ranges.

```
⟨*cp1255⟩
\DeclareInputText{130}{\quotesinglbase}
\DeclareInputText{131}{\textflorin}
\DeclareInputText{132}{\quotedblbase}
```

```
\DeclareInputText{133}{\dots}
\DeclareInputText{134}{\dag}
\DeclareInputText{135}{\ddag}
\DeclareInputText{136}{\^{}}
\DeclareInputText{137}{\textperthousand}
\DeclareInputText{139}{\guilsinglleft}
\DeclareInputText{145}{\textquoteleft}
\DeclareInputText{146}{\textquoteright}
\DeclareInputText{147}{\textquotedblleft}
\DeclareInputText{148}{\textquotedblright}
\DeclareInputText{149}{\textbullet}
\DeclareInputText{150}{\textendash}
\DeclareInputText{151}{\textemdash}
\DeclareInputText{152}{\~{}}
\DeclareInputText{153}{\texttrademark}
\DeclareInputText{155}{\guilsinglright}
⟨/cp1255⟩

⟨*8859 − 8 | cp1255⟩
\DeclareInputText{160}{\nobreakspace}
\DeclareInputText{162}{\textcent}
\DeclareInputText{163}{\pounds}
⟨+8859 − 8⟩\DeclareInputText{164}{\textcurrency}
⟨+cp1255⟩\DeclareInputText{164}{\newsheqel}
\DeclareInputText{165}{\textyen}
\DeclareInputText{166}{\textbrokenbar}
\DeclareInputText{167}{\S}
\DeclareInputText{168}{\"{}}
\DeclareInputText{169}{\textcopyright}
⟨+8859 − 8⟩\DeclareInputMath{170}{\times}
\DeclareInputText{171}{\guillemotleft}
\DeclareInputMath{172}{\lnot}
\DeclareInputText{173}{\-}
\DeclareInputText{174}{\textregistered}
\DeclareInputText{175}{\@tabacckludge={}}
\DeclareInputText{176}{\textdegree}
\DeclareInputMath{177}{\pm}
\DeclareInputMath{178}{\mathtwosuperior}
\DeclareInputMath{179}{\maththreesuperior}
\DeclareInputText{180}{\@tabacckludge'{}}
\DeclareInputMath{181}{\mu}
\DeclareInputText{182}{\P}
\DeclareInputText{183}{\textperiodcentered}
⟨+8859 − 8⟩\DeclareInputText{184}{\c\ }
\DeclareInputMath{185}{\mathonesuperior}
⟨+8859 − 8⟩\DeclareInputMath{186}{\div}
\DeclareInputText{187}{\guillemotright}
\DeclareInputText{188}{\textonequarter}
\DeclareInputText{189}{\textonehalf}
\DeclareInputText{190}{\textthreequarters}
```

```
⟨/8859 − 8 | cp1255⟩
```

Hebrew vowels and dots (nikud) are included only to MS Windows cp1255 page and start from the position 192.

```
⟨*cp1255⟩
\DeclareInputText{192}{\hebsheva}
\DeclareInputText{193}{\hebhatafsegol}
\DeclareInputText{194}{\hebhatafpatah}
\DeclareInputText{195}{\hebhatafqamats}
\DeclareInputText{196}{\hebhiriq}
\DeclareInputText{197}{\hebtsere}
\DeclareInputText{198}{\hebsegol}
\DeclareInputText{199}{\hebpatah}
\DeclareInputText{200}{\hebqamats}
\DeclareInputText{201}{\hebholam}
\DeclareInputText{203}{\hebqubuts}
\DeclareInputText{204}{\hebdagesh}
\DeclareInputText{205}{\hebmeteg}
\DeclareInputText{206}{\hebmaqaf}
\DeclareInputText{207}{\hebrafe}
\DeclareInputText{208}{\hebpaseq}
\DeclareInputText{209}{\hebshindot}
\DeclareInputText{210}{\hebsindot}
\DeclareInputText{211}{\hebsofpasuq}
\DeclareInputText{212}{\hebdoublevav}
\DeclareInputText{213}{\hebvavyod}
\DeclareInputText{214}{\hebdoubleyod}
⟨/cp1255⟩
```

Hebrew letters start from the position 224 in both encodings.

```
⟨*8859 − 8 | cp1255⟩
\DeclareInputText{224}{\hebalef}
\DeclareInputText{225}{\hebbet}
\DeclareInputText{226}{\hebgimel}
\DeclareInputText{227}{\hebdalet}
\DeclareInputText{228}{\hebhe}
\DeclareInputText{229}{\hebvav}
\DeclareInputText{230}{\hebzayin}
\DeclareInputText{231}{\hebhet}
\DeclareInputText{232}{\hebtet}
\DeclareInputText{233}{\hebyod}
\DeclareInputText{234}{\hebfinalkaf}
\DeclareInputText{235}{\hebkaf}
\DeclareInputText{236}{\heblamed}
\DeclareInputText{237}{\hebfinalmem}
\DeclareInputText{238}{\hebmem}
\DeclareInputText{239}{\hebfinalnun}
\DeclareInputText{240}{\hebnun}
\DeclareInputText{241}{\hebsamekh}
\DeclareInputText{242}{\hebayin}
```

```
\DeclareInputText{243}{\hebfinalpe}
\DeclareInputText{244}{\hebpe}
\DeclareInputText{245}{\hebfinaltsadi}
\DeclareInputText{246}{\hebtsadi}
\DeclareInputText{247}{\hebqof}
\DeclareInputText{248}{\hebresh}
\DeclareInputText{249}{\hebshin}
\DeclareInputText{250}{\hebtav}
⟨/8859 − 8 | cp1255⟩
```

Special symbols which define the direction of symbols explicitly. Currently, they are not used in LaTeX.

```
⟨*cp1255⟩
\DeclareInputText{253}{\lefttorightmark}
\DeclareInputText{254}{\righttoleftmark}
⟨/cp1255⟩
```

## 61.4 The IBM code page 862

The `cp862.def` encoding file defines the characters in the IBM codepage 862 encoding. The DOS graphics 'letters' and a few other positions are ignored (left undefined).

Hebrew letters start from the position 128.

```
⟨*cp862⟩
\DeclareInputText{128}{\hebalef}
\DeclareInputText{129}{\hebbet}
\DeclareInputText{130}{\hebgimel}
\DeclareInputText{131}{\hebdalet}
\DeclareInputText{132}{\hebhe}
\DeclareInputText{133}{\hebvav}
\DeclareInputText{134}{\hebzayin}
\DeclareInputText{135}{\hebhet}
\DeclareInputText{136}{\hebtet}
\DeclareInputText{137}{\hebyod}
\DeclareInputText{138}{\hebfinalkaf}
\DeclareInputText{139}{\hebkaf}
\DeclareInputText{140}{\heblamed}
\DeclareInputText{141}{\hebfinalmem}
\DeclareInputText{142}{\hebmem}
\DeclareInputText{143}{\hebfinalnun}
\DeclareInputText{144}{\hebnun}
\DeclareInputText{145}{\hebsamekh}
\DeclareInputText{146}{\hebayin}
\DeclareInputText{147}{\hebfinalpe}
\DeclareInputText{148}{\hebpe}
\DeclareInputText{149}{\hebfinaltsadi}
\DeclareInputText{150}{\hebtsadi}
\DeclareInputText{151}{\hebqof}
\DeclareInputText{152}{\hebresh}
```

```
\DeclareInputText{153}{\hebshin}
\DeclareInputText{154}{\hebtav}

\DeclareInputText{155}{\textcent}
\DeclareInputText{156}{\pounds}
\DeclareInputText{157}{\textyen}
\DeclareInputText{158}{\textpeseta}
\DeclareInputText{159}{\textflorin}
\DeclareInputText{160}{\@tabacckludge'a}
\DeclareInputText{161}{\@tabacckludge'\i}
\DeclareInputText{162}{\@tabacckludge'o}
\DeclareInputText{163}{\@tabacckludge'u}
\DeclareInputText{164}{\~n}
\DeclareInputText{165}{\~N}
\DeclareInputMath{166}{\mathordfeminine}
\DeclareInputMath{167}{\mathordmasculine}
\DeclareInputText{168}{\textquestiondown}
\DeclareInputMath{170}{\lnot}
\DeclareInputText{171}{\textonehalf}
\DeclareInputText{172}{\textonequarter}
\DeclareInputText{173}{\textexclamdown}
\DeclareInputText{174}{\guillemotleft}
\DeclareInputText{175}{\guillemotright}

\DeclareInputMath{224}{\alpha}
\DeclareInputText{225}{\ss}
\DeclareInputMath{226}{\Gamma}
\DeclareInputMath{227}{\pi}
\DeclareInputMath{228}{\Sigma}
\DeclareInputMath{229}{\sigma}
\DeclareInputMath{230}{\mu}
\DeclareInputMath{231}{\tau}
\DeclareInputMath{232}{\Phi}
\DeclareInputMath{233}{\Theta}
\DeclareInputMath{234}{\Omega}
\DeclareInputMath{235}{\delta}
\DeclareInputMath{236}{\infty}
\DeclareInputMath{237}{\phi}
\DeclareInputMath{238}{\varepsilon}
\DeclareInputMath{239}{\cap}
\DeclareInputMath{240}{\equiv}
\DeclareInputMath{241}{\pm}
\DeclareInputMath{242}{\ge}
\DeclareInputMath{243}{\le}
\DeclareInputMath{246}{\div}
\DeclareInputMath{247}{\approx}
\DeclareInputText{248}{\textdegree}
\DeclareInputText{249}{\textperiodcentered}
\DeclareInputText{250}{\textbullet}
\DeclareInputMath{251}{\surd}
\DeclareInputMath{252}{\mathnsuperior}
```

```
\DeclareInputMath{253}{\mathtwosuperior}
\DeclareInputText{254}{\textblacksquare}
\DeclareInputText{255}{\nobreakspace}
⟨/cp862⟩
```

\DisableNikud A utility macro to ignore any nikud character that may appear in the input. This allows you to ignore cp1255 nikud characters that happened to appear in the input.

```
⟨*8859 − 8⟩
\newcommand{\DisableNikud}{%
  \DeclareInputText{192}{}%
  \DeclareInputText{193}{}%
  \DeclareInputText{194}{}%
  \DeclareInputText{195}{}%
  \DeclareInputText{196}{}%
  \DeclareInputText{197}{}%
  \DeclareInputText{198}{}%
  \DeclareInputText{199}{}%
  \DeclareInputText{200}{}%
  \DeclareInputText{201}{}%
  \DeclareInputText{203}{}%
  \DeclareInputText{204}{}%
  \DeclareInputText{205}{}%
  \DeclareInputText{206}{}%
  \DeclareInputText{207}{}%
  \DeclareInputText{208}{}%
  \DeclareInputText{209}{}%
  \DeclareInputText{210}{}%
  \DeclareInputText{211}{}%
  \DeclareInputText{212}{}%
  \DeclareInputText{213}{}%
  \DeclareInputText{214}{}%
}
⟨/8859 − 8⟩
```

Finally, we reset the category code of the `@` sign at the end of all `.def` files.

```
⟨−driver⟩\makeatother
```

## 62 Hebrew font encodings

Don't forget to update the docs...

### 62.1 THIS SECTION IS OUT OF DATE. UPDATE DOCS TO MATCH HE8 ENCODING

The file `hebrew.fdd`[73] contains the Local Hebrew Encoding (`LHE`) definition, the external font information needed to use the Hebrew 7-bit fonts (old code fonts)

[73]The files described in this section have version number v1.2c and were last revised on 2005/05/20.

and `hebfont` package that provides Hebrew font switching commands.

Using this file as an input, `lheenc.def` encoding definition file, all `.fd` files (font definition files) and font switching package for available Hebrew fonts are generated. We chose to use 7-bit encoding as default font encoding, because:

1. There are many 7-bit encoded Hebrew fonts available, more then for any other encoding.

2. Available TEX Hebrew fonts do not include latin alphabet, and we can safely map Hebrew glyphs to the `ASCII` positions (0 – 127).

Current definition of the `LHE` encoding supports only Hebrew letters (`\hebalef`–`\hebtav`), but not Hebrew points, such as `\hebdagesh`, `\hebqamats`, `\hebpatah`, `\hebshindot`, etc. We are working now on such addition.

## 62.2 The DOCSTRIP modules

The following modules are used in the implementation to direct DOCSTRIP in generating external files:

| | |
|---|---|
| driver | produce a documentation driver file |
| HE8enc | produce the encoding definition for CodePage 1255 (`HE8`) |
| HE8cmr | make Hebrew default font in `HE8` |
| HE8cmss | make Hebrew sans-serif font in `HE8` |
| HE8cmtt | make Hebrew typewriter font in `HE8` |
| HE8OmegaHebrew | Hebrew font from the Omega project (by ???) |
| HE8aharoni | Hebrew sans-serif font (Culmus) |
| HE8david | Hebrew serif font (Culmus) |
| HE8drugulin | Hebrew old serif font (Culmus) |
| HE8ellinia | Hebrew isans-serif font (Culmus) |
| HE8frankruehl | Hebrew serif font (Culmus) |
| HE8KtavYad | Hebrew handwriting font (Culmus) |
| HE8MiriamMono | Hebrew monospaced font |
| HE8Nachlieli | Hebrew sans-serif font (Culmus) |
| HE8CourierShalom | Hebrew Shalom (Courier) font (by IBM) |
| HE8HelveticaNarkissTam | Hebrew NarkisTam (Helvetica) (by Zvi Narkis) |
| HE8TimesNarkissim | Hebrew Narkissim (Times) (by Zvi Narkis) |
| HE8mfdavid | Hebrew David font (by ???) |
| HE8mffrank | Hebrew Frank-Ruehl font (by ??) |
| HE8mffrankthick | Hebrew Frank-Ruehl (thick) font (by ??) |
| HE8mffrankthin | Hebrew Frank-Ruehl (thin) font (by ??) |
| HE8mfmiriam | Hebrew Miriam font (by ???) |
| HE8mfmiriamwide | Hebrew Miriam (wide) font (by ???) |
| HE8mfnarkistam | Hebrew Narkis Tam font (by ???) |
| LHEenc | produce the encoding definition for Local Hebrew Encoding (`LHE`) |
| LHEcmr | make Hebrew default font in `LHE` |
| LHEcmss | make Hebrew sans-serif font in `LHE` |
| LHEcmtt | make Hebrew typewriter font in `LHE` |
| LHEclas | make Hebrew classic font (by Joel M. Hoffman) in `LHE` |
| LHEshold | make Hebrew shalom old font (by Jonathan Brecher) in `LHE` |
| LHEshscr | make Hebrew shalom script font (by Jonathan Brecher) in `LHE` |
| LHEshstk | make Hebrew shalom stick font (by Jonathan Brecher) in `LHE` |
| LHEfr | make Hebrew frank-ruehl font in `LHE` |
| LHEcrml | make Hebrew carmel font (by Dr. Samy Zafrany) in `LHE` |
| LHEredis | make Hebrew redis font (by Prof. Jacques J. Goldberg) in `LHE` |
| nowarn | option for font definition files, that used to produce "silent" font substitutions without giving warnings |
| hebfont | create Hebrew font switching commands package |

A typical DOCSTRIP command file would then have entries like:

```
\generateFile{lhecmr.fd}{t}{\from{hebrew.fdd}{LHEcmr,nowarn}}
```

## 62.3 The LHEencoding definition file

The Hebrew font encoding LHE is based upon the old-code encoding also known as the Israeli Standard SI-960. Many Hebrew TEX fonts from the Hebrew University of Jerusalem are encoded in this encoding. It only uses the lower 128 positions of the font table. As local encoding its name start with the letter ‘L’.

First we define the Local Hebrew Encoding; specify a default for the font substitution process for the LHE encoding and supply a font to be used when all else fails.

```
⟨*LHEenc⟩
\DeclareFontEncoding{LHE}{}{}
\DeclareFontSubstitution{LHE}{cmr}{m}{n}
\DeclareErrorFont{LHE}{cmr}{m}{n}{10}
⟨/LHEenc⟩
```

Then we define a few commands in the LHE encoding.

```
⟨*LHEenc⟩
\ProvideTextCommand{\textcopyright}{LHE}{\textcircled{\@latin{c}}}
\ProvideTextCommand{\textregistered}{LHE}{\textcircled{\scshape%
                                                 \@latin{r}}}
\ProvideTextCommand{\texttrademark}{LHE}{\textsuperscript{\@latin{TM}}}
⟨/LHEenc⟩
```

Because not everyone can input Hebrew input text directly from the keyboard we need to define control sequences for all the Hebrew glyphs in the fonts. In addition, we want to support many input encodings for Hebrew and to keep the language definition file (hebrew.ldf) independent of the encoding. Therefore, we exploit the standard LaTeX 2$_\varepsilon$ font encoding mechanism to define control sequences for all the Hebrew glyphs in the fonts in encoding-specific way. The language definition file uses only the control sequences and doesn't need to check the current font or input encoding.

In the LHE encoding (7-bit encoding) all the Hebrew glyphes reside in the *lower* half of the font. Currently, only the Hebrew letters are supported. They use the same positions as the latin small letters in ASCII encoding and the position of `.

The symbol ` (glyph 96) is used by Hebrew letter *Alef*, so we need to define its lccode to allow hyphenation. All other letters retain the same lccodes as their latin counterparts.

```
⟨+LHEenc⟩\lccode``=``
```

Hebrew letters occupy the positions 96–122 in LHE encoding:

```
⟨*LHEenc⟩
\DeclareTextSymbol{\hebalef}{LHE}{96}
\DeclareTextSymbol{\hebbet}{LHE}{97}
\DeclareTextSymbol{\hebgimel}{LHE}{98}
\DeclareTextSymbol{\hebdalet}{LHE}{99}
\DeclareTextSymbol{\hebhe}{LHE}{100}
\DeclareTextSymbol{\hebvav}{LHE}{101}
\DeclareTextSymbol{\hebzayin}{LHE}{102}
\DeclareTextSymbol{\hebhet}{LHE}{103}
```

```
\DeclareTextSymbol{\hebtet}{LHE}{104}
\DeclareTextSymbol{\hebyod}{LHE}{105}
\DeclareTextSymbol{\hebfinalkaf}{LHE}{106}
\DeclareTextSymbol{\hebkaf}{LHE}{107}
\DeclareTextSymbol{\heblamed}{LHE}{108}
\DeclareTextSymbol{\hebfinalmem}{LHE}{109}
\DeclareTextSymbol{\hebmem}{LHE}{110}
\DeclareTextSymbol{\hebfinalnun}{LHE}{111}
\DeclareTextSymbol{\hebnun}{LHE}{112}
\DeclareTextSymbol{\hebsamekh}{LHE}{113}
\DeclareTextSymbol{\hebayin}{LHE}{114}
\DeclareTextSymbol{\hebfinalpe}{LHE}{115}
\DeclareTextSymbol{\hebpe}{LHE}{116}
\DeclareTextSymbol{\hebfinaltsadi}{LHE}{117}
\DeclareTextSymbol{\hebtsadi}{LHE}{118}
\DeclareTextSymbol{\hebqof}{LHE}{119}
\DeclareTextSymbol{\hebresh}{LHE}{120}
\DeclareTextSymbol{\hebshin}{LHE}{121}
\DeclareTextSymbol{\hebtav}{LHE}{122}
⟨/LHEenc⟩
```

Letter `\hebsin` is defined as a synonym of `\hebshin`:

```
⟨+LHEenc⟩\let\hebsin=\hebshin
```

## 62.4 The font definition files (in LHE encoding)

### 62.4.1 Hebrew default font

It uses *Jerusalem* font for regular font, *Old Jaffa* font for italic shape and small-caps, *Dead Sea* font for bold face, and *Tel-Aviv* for bold-italic

```
⟨*LHEcmr⟩
\DeclareFontFamily{LHE}{cmr}{\hyphenchar\font45}
\DeclareFontShape{LHE}{cmr}{m}{n}
      {<-> jerus10 }{}
%%%%%%% Italicized shape
\DeclareFontShape{LHE}{cmr}{m}{it}
      {<-> oldjaf10 }{}
\DeclareFontShape{LHE}{cmr}{m}{sl}
      {<-> oldjaf10 }{}
\DeclareFontShape{LHE}{cmr}{m}{sc}
      {<-> oldjaf10 }{}
%%%%%%% Bold extended series
\DeclareFontShape{LHE}{cmr}{bx}{n}
      {<-> deads10 }{}
\DeclareFontShape{LHE}{cmr}{b}{n}
      {<-> deads10 }{}
%%%%%%% Bold extended (Italic)  series
\DeclareFontShape{LHE}{cmr}{bx}{sl}
      {<-> telav10 }{}
\DeclareFontShape{LHE}{cmr}{bx}{it}
```

```
      {<-> telav10 }{}
⟨/LHEcmr⟩
```

### 62.4.2 Hebrew sans-serif font

We use *Tel Aviv* font for the Sans family. *Old Jaffa* font is used for italic shape and *Dead Sea* used for bold face.

```
⟨*LHEcmss⟩
\DeclareFontFamily{LHE}{cmss}{\hyphenchar\font45}
\DeclareFontShape{LHE}{cmss}{m}{n}
      {<-> telav10 }{}
%%%%%%% Font/shape undefined, therefore substituted
\DeclareFontShape{LHE}{cmss}{m}{sc}
⟨−nowarn⟩  {<->sub * cmss/m/n}{}
⟨+nowarn⟩  {<->ssub * cmss/m/n}{}
%%%%%%% Italicized shape
\DeclareFontShape{LHE}{cmss}{m}{it}
      {<-> oldjaf10 }{}
%%%%%%% Font/shape undefined, therefore substituted
\DeclareFontShape{LHE}{cmss}{m}{sl}
⟨−nowarn⟩  {<->sub * cmss/m/it}{}
⟨+nowarn⟩  {<->ssub * cmss/m/it}{}
%%%%%%% Bold extended series
\DeclareFontShape{LHE}{cmss}{bx}{n}
      {<-> deads10 }{}
%%%%%%% Font/shape undefined, therefore substituted
\DeclareFontShape{LHE}{cmss}{b}{n}
⟨−nowarn⟩  {<->sub * cmss/bx/n}{}
⟨+nowarn⟩  {<->ssub * cmss/bx/n}{}
%%%%%%% Font/shape undefined, therefore substituted
\DeclareFontShape{LHE}{cmss}{bx}{sl}
⟨−nowarn⟩  {<->sub * cmss/bx/n}{}
⟨+nowarn⟩  {<->ssub * cmss/bx/n}{}
%%%%%%% Font/shape undefined, therefore substituted
\DeclareFontShape{LHE}{cmss}{bx}{it}
⟨−nowarn⟩  {<->sub * cmss/bx/n}{}
⟨+nowarn⟩  {<->ssub * cmss/bx/n}{}
⟨/LHEcmss⟩
```

### 62.4.3 Hebrew typewriter font

We use *Tel Aviv* font as the typewriter font. *Old Jaffa* font is used for italic shape and *Dead Sea* used for bold face.

```
⟨*LHEcmtt⟩
\DeclareFontFamily{LHE}{cmtt}{\hyphenchar \font\m@ne}
\DeclareFontShape{LHE}{cmtt}{m}{n}
      {<-> telav10 }{}
%%%%%%% Font/shape undefined, therefore substituted
\DeclareFontShape{LHE}{cmtt}{m}{sc}
```

```
⟨−nowarn⟩  {<->sub * cmtt/m/n}{}
⟨+nowarn⟩  {<->ssub * cmtt/m/n}{}
%%%%%%% Italicized shape
\DeclareFontShape{LHE}{cmtt}{m}{it}
      {<-> oldjaf10 }{}
%%%%%%% Font/shape undefined, therefore substituted
\DeclareFontShape{LHE}{cmtt}{m}{sl}
⟨−nowarn⟩  {<->sub * cmtt/m/it}{}
⟨+nowarn⟩  {<->ssub * cmtt/m/it}{}
%%%%%%% Bold extended series
\DeclareFontShape{LHE}{cmtt}{bx}{n}
      {<-> deads10 }{}
%%%%%%% Font/shape undefined, therefore substituted
\DeclareFontShape{LHE}{cmtt}{bx}{it}
⟨−nowarn⟩  {<->sub * cmtt/bx/n}{}
⟨+nowarn⟩  {<->ssub * cmtt/bx/n}{}
⟨/LHEcmtt⟩
```

### 62.4.4 Hebrew classic font

*Hclassic* and *hcaption* fonts are distributed freely from `CTAN` sites and copyrighted by Joel M. Hoffman, of 19 Hillcrest Lane, Rye, NY 10580 USA, e-mail: `72700.402@compuserve.com`.

Hclassic is a modernized Classical Hebrew font (in the same way that Knuth's `cmr` family is a modernized Roman font — but his fonts are much nicer). Hcaption is a slanted version of hclassic font. Both fonts contain all of the Hebrew consonants, the (rarely used) ligature *alef-lamed* and two versions of the letter *ayin* for use with and without vowels. Hclassic also contains all of the vowels found in Hebrew, a symbol for *meteg*, and dots for use as a *dagesh* and for differentiating *shin* and *sin* letters.

Currently, only the Hebrew consonants (*hebalef* – *hebtav*) from these fonts are supported by LaTeX 2$_\varepsilon$, however one can use vowels and dots directly with Plain TeX macros. We are working on generic vowels and dots support for LaTeX 2$_\varepsilon$.

```
⟨*LHEclas⟩
\DeclareFontFamily{LHE}{clas}{}
\DeclareFontShape{LHE}{clas}{m}{n}
      {<-> s * [0.83345] hclassic }{}
%%%%%%% Font/shape undefined, therefore substituted
\DeclareFontShape{LHE}{clas}{m}{sc}
⟨−nowarn⟩  {<->sub * clas/m/n}{}
⟨+nowarn⟩  {<->ssub * clas/m/n}{}
%%%%%%% Slanted shape
\DeclareFontShape{LHE}{clas}{m}{sl}
      {<-> s * [0.69389] hcaption }{}
%%%%%%% Font/shape undefined, therefore substituted
\DeclareFontShape{LHE}{clas}{m}{it}
⟨−nowarn⟩  {<->sub * clas/m/sl}{}
```

```
⟨+nowarn⟩   {<->ssub * clas/m/sl}{}
⟨/LHEclas⟩
```

### 62.4.5 Hebrew shalom fonts

All three shalom fonts (*ShalomScript10*, *ShalomStick10* and *ShalomOldStyle10*) have been created by Jonathan Brecher, of 9 Skyview Road, Lexington, MA 02173-1112 USA, e-mail: `brecher@husc.harvard.edu`.

All shalom fonts have been written in PostScript via Fontographer on a Mac. The fonts have been converted to METAFONT by Rama Porrat (e-mail: `rama@cc.huji.ac.il`), using the utility typo, a font editor + converter between font formats (a commercial product). `ShalomScript10.mf` is the METAFONT equivalent of `ShalomScript.ps`, `ShalomStick10.mf` came from `ShalomStick.ps` and `ShalomOldStyle10.mf` originated in `ShalomOldStyle.ps`.

The fonts differ in the letters' style. ShalomScript10 contains hand writing Hebrew letters; ShalomStick10 contains sans-serif letters, and ShalomOldStyle10 contains old style letters. All three fonts contain vowels and dots (nikud). While converting to METAFONT, letters and symbols within the fonts have been arranged so as to get a usable font for writing Hebrew documents in TeX or LaTeX, with as well as without vowels.

Currently, only the Hebrew consonants (*hebalef – hebtav*) from these fonts are supported by LaTeX $2_\varepsilon$, however one can use vowels and dots directly with Plain TeX macros. We are working on generic vowels and dots support for LaTeX $2_\varepsilon$.

```
⟨*LHEshold⟩
\DeclareFontFamily{LHE}{shold}{}
\DeclareFontShape{LHE}{shold}{m}{n}
      {<-> shold10 }{}
⟨/LHEshold⟩
⟨*LHEshscr⟩
\DeclareFontFamily{LHE}{shscr}{}
\DeclareFontShape{LHE}{shscr}{m}{n}
      {<-> shscr10 }{}
⟨/LHEshscr⟩
⟨*LHEshstk⟩
\DeclareFontFamily{LHE}{shstk}{}
\DeclareFontShape{LHE}{shstk}{m}{n}
      {<-> shstk10 }{}
⟨/LHEshstk⟩
```

### 62.4.6 Hebrew frank-ruehl font

*Frank Ruehl* font was written in METAFONT and includes three shapes: regular, bold extaneded and slanted.

```
⟨*LHEfr⟩
\DeclareFontFamily{LHE}{fr}{}
\DeclareFontShape{LHE}{fr}{m}{n}
```

```
      {<-> fr }{}
%%%%%%% Font/shape undefined, therefore substituted
\DeclareFontShape{LHE}{fr}{m}{sc}
⟨−nowarn⟩  {<->sub * fr/m/n}{}
⟨+nowarn⟩  {<->ssub * fr/m/n}{}
%%%%%%% Slanted shape
\DeclareFontShape{LHE}{fr}{m}{sl}
      {<-> frsl }{}
%%%%%%% Font/shape undefined, therefore substituted
\DeclareFontShape{LHE}{fr}{m}{it}
⟨−nowarn⟩  {<->sub * fr/m/sl}{}
⟨+nowarn⟩  {<->ssub * fr/m/sl}{}
%%%%%%% Bold extended series
\DeclareFontShape{LHE}{fr}{bx}{n}
      {<-> frbx }{}
%%%%%%% Font/shape undefined, therefore substituted
\DeclareFontShape{LHE}{fr}{b}{n}
⟨−nowarn⟩  {<->sub * fr/bx/n}{}
⟨+nowarn⟩  {<->ssub * fr/bx/n}{}
%%%%%%% Font/shape undefined, therefore substituted
\DeclareFontShape{LHE}{fr}{bx}{sl}
⟨−nowarn⟩  {<->sub * fr/bx/n}{}
⟨+nowarn⟩  {<->ssub * fr/bx/n}{}
%%%%%%% Font/shape undefined, therefore substituted
\DeclareFontShape{LHE}{fr}{bx}{it}
⟨−nowarn⟩  {<->sub * fr/bx/n}{}
⟨+nowarn⟩  {<->ssub * fr/bx/n}{}
⟨/LHEfr⟩
```

### 62.4.7 Hebrew carmel font

*Carmel* font includes regular and slanted shapes. It was created by Dr. Samy Zafrany of the Technion, Haifa, Israel with the intention of making nice fonts for headers and emphasized text.

```
⟨*LHEcrml⟩
\DeclareFontFamily{LHE}{crml}{}
\DeclareFontShape{LHE}{crml}{m}{n}
      {<-> crml10 }{}
%%%%%%% Font/shape undefined, therefore substituted
\DeclareFontShape{LHE}{crml}{m}{sc}
⟨−nowarn⟩  {<->sub * crml/m/n}{}
⟨+nowarn⟩  {<->ssub * crml/m/n}{}
%%%%%%% Slanted shape
\DeclareFontShape{LHE}{crml}{m}{sl}
      {<-> crmlsl10 }{}
%%%%%%% Font/shape undefined, therefore substituted
\DeclareFontShape{LHE}{crml}{m}{it}
⟨−nowarn⟩  {<->sub * crml/m/sl}{}
⟨+nowarn⟩  {<->ssub * crml/m/sl}{}
```

```
⟨/LHEcrml⟩
```

### 62.4.8 Hebrew redis font

*Redis* font has been created by Prof. Jacques J. Goldberg of the Technion. Haifa, Israel. The font is available in regular, slanted and bold extanded shapes. This font contains a full set of Hebrew letters in a "sans-serif vectorized" style, and selected punctuation.

```
⟨*LHEredis⟩
\DeclareFontFamily{LHE}{redis}{}
\DeclareFontShape{LHE}{redis}{m}{n}{%
  <5> <6> redis7
  <7> <8> <9> <10> <12> gen * redis
  <10.95> redis10
  <14.4> redis12
  <17.28> <20.74> <24.88> redis17}{}
%%%%%%% Font/shape undefined, therefore substituted
\DeclareFontShape{LHE}{redis}{m}{sc}
⟨−nowarn⟩  {<->sub * redis/m/n}{}
⟨+nowarn⟩  {<->ssub * redis/m/n}{}
%%%%%%% Slanted shape
\DeclareFontShape{LHE}{redis}{m}{sl}{%
  <5> <6> <7> rediss8
  <8> <9> <10> <12> gen * rediss
  <10.95> rediss10
  <14.4> <17.28> <20.74> <24.88> rediss12}{}
%%%%%%% Font/shape undefined, therefore substituted
\DeclareFontShape{LHE}{redis}{m}{it}
⟨−nowarn⟩  {<->sub * redis/m/sl}{}
⟨+nowarn⟩  {<->ssub * redis/m/sl}{}
%%%%%%% Bold extended series
\DeclareFontShape{LHE}{redis}{bx}{n}{%
  <5> <6> <7> <8> <9> <10> <10.95> <12>
  <14.4> <17.28> <20.74> <24.88> redisb10}{}
%%%%%%% Font/shape undefined, therefore substituted
\DeclareFontShape{LHE}{redis}{b}{n}
⟨−nowarn⟩  {<->sub * redis/bx/n}{}
⟨+nowarn⟩  {<->ssub * redis/bx/n}{}
%%%%%%% Font/shape undefined, therefore substituted
\DeclareFontShape{LHE}{redis}{bx}{sl}
⟨−nowarn⟩  {<->sub * redis/bx/n}{}
⟨+nowarn⟩  {<->ssub * redis/bx/n}{}
%%%%%%% Font/shape undefined, therefore substituted
\DeclareFontShape{LHE}{redis}{bx}{it}
⟨−nowarn⟩  {<->sub * redis/bx/n}{}
⟨+nowarn⟩  {<->ssub * redis/bx/n}{}
⟨/LHEredis⟩
```

## 62.5 The `HE8`encoding definition file

The Hebrew font encoding `HE8` is based upon an extention by Microsoft to the ISO-8859-8 standard. This is an 8bit encoding. The extentions include hebrew points ("Nikud").

First we define the Codepage 1255; specify a default for the font substitution process for the `HE8` encoding and supply a font to be used when all else fails.

```
⟨*HE8enc⟩
\DeclareFontEncoding{HE8}{}{}
\DeclareFontSubstitution{HE8}{cmr}{m}{n}
\DeclareErrorFont{HE8}{cmr}{m}{n}{10}
⟨/HE8enc⟩
```

Then we define a few commands in the `HE8` encoding.

```
⟨*HE8enc⟩
\ProvideTextCommand{\textcopyright}{HE8}{\textcircled{\@latin{c}}}
\ProvideTextCommand{\textregistered}{HE8}{\textcircled{\scshape%
                                         \@latin{r}}}
\ProvideTextCommand{\texttrademark}{HE8}{\textsuperscript{\@latin{TM}}}
⟨/HE8enc⟩
```

### 62.5.1 CHECK HERE FOR HE8 UPDATES

Because not everyone can input Hebrew input text directly from the keyboard we need to define control sequences for all the Hebrew glyphs in the fonts. In addition, we want to support many input encodings for Hebrew and to keep the language definition file (`hebrew.ldf`) independent of the encoding. Therefore, we exploit the standard LaTeX 2ε font encoding mechanism to define control sequences for all the Hebrew glyphs in the fonts in encoding-specific way. The language definition file uses only the control sequences and doesn't need to check the current font or input encoding.

In the `LHE` encoding (7-bit encoding) all the Hebrew glyphes reside in the *lower* half of the font. Currently, only the Hebrew letters are supported. They use the same positions as the latin small letters in `ASCII` encoding and the position of `‘`.

Some general symbols:

```
⟨*HE8enc⟩
\ProvideTextCommand{\textcopyright}{HE8}{\textcircled{\@latin{c}}}
\ProvideTextCommand{\textregistered}{HE8}{\textcircled{\scshape%
                                         \@latin{r}}}
\ProvideTextCommand{\texttrademark}{HE8}{\textsuperscript{\@latin{TM}}}
⟨/HE8enc⟩
```

The hebrew points:

```
⟨*HE8enc⟩
\DeclareTextSymbol{\sheva}{HE8}{192}
\DeclareTextSymbol{\hatafsegol}{HE8}{193}
\DeclareTextSymbol{\hatafpatah}{HE8}{194}
\DeclareTextSymbol{\hatafqamats}{HE8}{195}
\DeclareTextSymbol{\hiriq}{HE8}{196}
```

```
\DeclareTextSymbol{\tsere}{HE8}{197}
\DeclareTextSymbol{\segol}{HE8}{198}
\DeclareTextSymbol{\patah}{HE8}{199}
\DeclareTextSymbol{\qamats}{HE8}{200}
\DeclareTextSymbol{\holam}{HE8}{201}
\DeclareTextSymbol{\qubuts}{HE8}{203}
\DeclareTextSymbol{\dagesh}{HE8}{204}
\DeclareTextSymbol{\meteg}{HE8}{205}
\DeclareTextSymbol{\maqaf}{HE8}{206}
\DeclareTextSymbol{\rafe}{HE8}{207}
\DeclareTextSymbol{\paseq}{HE8}{208}
\DeclareTextSymbol{\shindot}{HE8}{209}
\DeclareTextSymbol{\sindot}{HE8}{210}
\DeclareTextSymbol{\sofpasuq}{HE8}{211}
\DeclareTextSymbol{\doublevav}{HE8}{212}
\DeclareTextSymbol{\vavyod}{HE8}{213}
\DeclareTextSymbol{\doubleyod}{HE8}{214}
⟨/HE8enc⟩
```

Hebrew letters occupy the positions 224–250 in HE8 encoding [WHAT ABOUT OTHER MARKS]:

```
⟨*HE8enc⟩
% \lccode``=`` % probably not needed (Tzafrir)
\DeclareTextSymbol{\hebalef}{HE8}{224}
\DeclareTextSymbol{\hebbet}{HE8}{225}
\DeclareTextSymbol{\hebgimel}{HE8}{226}
\DeclareTextSymbol{\hebdalet}{HE8}{227}
\DeclareTextSymbol{\hebhe}{HE8}{228}
\DeclareTextSymbol{\hebvav}{HE8}{229}
\DeclareTextSymbol{\hebzayin}{HE8}{230}
\DeclareTextSymbol{\hebhet}{HE8}{231}
\DeclareTextSymbol{\hebtet}{HE8}{232}
\DeclareTextSymbol{\hebyod}{HE8}{233}
\DeclareTextSymbol{\hebfinalkaf}{HE8}{234}
\DeclareTextSymbol{\hebkaf}{HE8}{235}
\DeclareTextSymbol{\heblamed}{HE8}{236}
\DeclareTextSymbol{\hebfinalmem}{HE8}{237}
\DeclareTextSymbol{\hebmem}{HE8}{238}
\DeclareTextSymbol{\hebfinalnun}{HE8}{239}
\DeclareTextSymbol{\hebnun}{HE8}{240}
\DeclareTextSymbol{\hebsamekh}{HE8}{241}
\DeclareTextSymbol{\hebayin}{HE8}{242}
\DeclareTextSymbol{\hebfinalpe}{HE8}{243}
\DeclareTextSymbol{\hebpe}{HE8}{244}
\DeclareTextSymbol{\hebfinaltsadi}{HE8}{245}
\DeclareTextSymbol{\hebtsadi}{HE8}{246}
\DeclareTextSymbol{\hebqof}{HE8}{247}
\DeclareTextSymbol{\hebresh}{HE8}{248}
\DeclareTextSymbol{\hebshin}{HE8}{249}
\DeclareTextSymbol{\hebtav}{HE8}{250}
```

```
⟨/HE8enc⟩
```

Letter `\hebsin` is defined as a synonym of `\hebshin`:

```
⟨+HE8enc⟩\let\hebsin=\hebshin
```

## 62.6 The font definition files (in HE8 encoding)

### 62.6.1 Hebrew default font

It uses *OmegaHebrew* font for regular font, *Old Jaffa* font for italic shape and small-caps, *Dead Sea* font for bold face, and *Tel-Aviv* for bold-italic

```
⟨*HE8cmr⟩
\DeclareFontFamily{HE8}{cmr}{\hyphenchar\font45}
\DeclareFontShape{HE8}{cmr}{m}{n}
      {<-> david }{}
%%%%%%% Italicized shape
\DeclareFontShape{HE8}{cmr}{m}{it}
      {<-> davidi }{}
\DeclareFontShape{HE8}{cmr}{m}{sl}
      {<-> davidi }{}
\DeclareFontShape{HE8}{cmr}{m}{sc}
      {<-> david }{}
%%%%%%% Bold extended series
\DeclareFontShape{HE8}{cmr}{bx}{n}
      {<-> davidb }{}
\DeclareFontShape{HE8}{cmr}{b}{n}
      {<-> davidb }{}
%%%%%%% Bold extended (Italic)  series
\DeclareFontShape{HE8}{cmr}{bx}{sl}
      {<-> davidbi }{}
\DeclareFontShape{HE8}{cmr}{bx}{it}
      {<-> davidbi }{}
⟨/HE8cmr⟩
```

### 62.6.2 Hebrew sans-serif font

Until we have a real sans-serif font in this distribution, this file will remain a copy of the roman fonts definitons above.

```
⟨*HE8cmss⟩
\DeclareFontFamily{HE8}{cmss}{\hyphenchar\font45}
\DeclareFontShape{HE8}{cmss}{m}{n}
      {<-> nachlieli }{}
%%%%%%% Italicized shape
\DeclareFontShape{HE8}{cmss}{m}{it}
      {<-> nachlieli }{}
\DeclareFontShape{HE8}{cmss}{m}{sl}
      {<-> nachlieli }{}
\DeclareFontShape{HE8}{cmss}{m}{sc}
      {<-> nachlieli }{}
```

```
%%%%%%% Bold extended series
\DeclareFontShape{HE8}{cmss}{bx}{n}
      {<-> nachlieli }{}
\DeclareFontShape{HE8}{cmss}{b}{n}
      {<-> nachlieli }{}
%%%%%%% Bold extended (Italic)  series
\DeclareFontShape{HE8}{cmss}{bx}{sl}
      {<-> nachlieli }{}
\DeclareFontShape{HE8}{cmss}{bx}{it}
      {<-> nachlieli }{}
⟨/HE8cmss⟩
```

### 62.6.3 Hebrew typewriter font

Until we have a real sans-serif font in this distribution, this file will remain a copy of the roman fonts definitons above.

```
⟨*HE8cmtt⟩
\DeclareFontFamily{HE8}{cmtt}{\hyphenchar\font45}
\DeclareFontShape{HE8}{cmtt}{m}{n}
      {<-> miriam }{}
%%%%%%% Italicized shape
\DeclareFontShape{HE8}{cmtt}{m}{it}
      {<-> miriam }{}
\DeclareFontShape{HE8}{cmtt}{m}{sl}
      {<-> miriam }{}
\DeclareFontShape{HE8}{cmtt}{m}{sc}
      {<-> miriam }{}
%%%%%%% Bold extended series
\DeclareFontShape{HE8}{cmtt}{bx}{n}
      {<-> miriam }{}
\DeclareFontShape{HE8}{cmtt}{b}{n}
      {<-> miriam }{}
%%%%%%% Bold extended (Italic)  series
\DeclareFontShape{HE8}{cmtt}{bx}{sl}
      {<-> miriam }{}
\DeclareFontShape{HE8}{cmtt}{bx}{it}
      {<-> miriam }{}
⟨/HE8cmtt⟩
```

### 62.6.4 8Bit OmegaHebrew font

*OmegaHebrew* is a serif hebrew font created by the omega project [FILL IN CREDITS] [FILL IN GENERAL SHAPE DESCRIPTION] shapes: [FILL IN]

```
⟨*HE8OmegaHebrew⟩
\def\OmegaHebrewscale{0.9}
\DeclareFontFamily{HE8}{OmegaHebrew}{\hyphenchar\font45}
\DeclareFontShape{HE8}{OmegaHebrew}{m}{n}{<-> [\OmegaHebrewscale] OmegaHebrew }{}
%\endinput % is it needed [tzafrir]
⟨/HE8OmegaHebrew⟩
```

### 62.6.5 8Bit Aharoni font

*Aharoni* is a serif hebrew font created by the omega project [FILL IN CREDITS] [FILL IN GENERAL SHAPE DESCRIPTION] shapes: [FILL IN]

```
⟨*HE8aharoni⟩
\def\Aharoniscale{1.0}
\DeclareFontFamily{HE8}{aharoni}{\hyphenchar\font45}
\DeclareFontShape{HE8}{aharoni}{m}{n}   {<-> [\Aharoniscale] aharoni}{}
\DeclareFontShape{HE8}{aharoni}{m}{it}  {<-> [\Aharoniscale] aharonii}{}
\DeclareFontShape{HE8}{aharoni}{m}{sl}  {<-> [\Aharoniscale] aharonii}{}
\DeclareFontShape{HE8}{aharoni}{b}{n}   {<-> [\Aharoniscale] aharonib}{}
\DeclareFontShape{HE8}{aharoni}{bx}{n}  {<-> [\Aharoniscale] aharonib}{}
\DeclareFontShape{HE8}{aharoni}{bx}{it} {<-> [\Aharoniscale] aharonibi}{}

%\endinput % is it needed [tzafrir]
⟨/HE8aharoni⟩
```

### 62.6.6 8Bit David font

*David* is a serif hebrew font created by the omega project [FILL IN CREDITS] [FILL IN GENERAL SHAPE DESCRIPTION] shapes: [FILL IN]

```
⟨*HE8david⟩
\def\Davidscale{1.0}
\DeclareFontFamily{HE8}{david}{\hyphenchar\font45}

\DeclareFontShape{HE8}{david}{m}{n}   {<-> [\Davidscale] david}{}
\DeclareFontShape{HE8}{david}{m}{it}  {<-> [\Davidscale] davidi}{}
\DeclareFontShape{HE8}{david}{m}{sl}  {<-> [\Davidscale] davidi}{}
\DeclareFontShape{HE8}{david}{b}{n}   {<-> [\Davidscale] davidb}{}
\DeclareFontShape{HE8}{david}{bx}{n}  {<-> [\Davidscale] davidb}{}
\DeclareFontShape{HE8}{david}{bx}{it} {<-> [\Davidscale] davidbi}{}


%\endinput % is it needed [tzafrir]
⟨/HE8david⟩
```

### 62.6.7 8Bit Drugulin font

*Drugulin* is a serif hebrew font created by the omega project [FILL IN CREDITS] [FILL IN GENERAL SHAPE DESCRIPTION] shapes: [FILL IN]

```
⟨*HE8drugulin⟩
\def\Drugulinscale{1.0}
\DeclareFontFamily{HE8}{drugulin}{\hyphenchar\font45}
\DeclareFontShape{HE8}{drugulin}{m}{n}   {<-> [\Drugulinscale] drugulinb}{}
\DeclareFontShape{HE8}{drugulin}{m}{it}  {<-> [\Drugulinscale] drugulinbi}{}
\DeclareFontShape{HE8}{drugulin}{m}{sl}  {<-> [\Drugulinscale] drugulinbi}{}
\DeclareFontShape{HE8}{drugulin}{b}{n}   {<-> [\Drugulinscale] drugulinb}{}
\DeclareFontShape{HE8}{drugulin}{bx}{n}  {<-> [\Drugulinscale] drugulinb}{}
```

```
\DeclareFontShape{HE8}{drugulin}{bx}{it} {<-> [\Drugulinscale] drugulinbi}{}
%\endinput % is it needed [tzafrir]
⟨/HE8drugulin⟩
```

### 62.6.8 8Bit Ellinia font

*Ellinia* is a sans-serif hebrew font created by the omega project [FILL IN CREDITS] [FILL IN GENERAL SHAPE DESCRIPTION] shapes: [FILL IN]

```
⟨*HE8ellinia⟩
\def\Elliniascale{1.0}
\DeclareFontFamily{HE8}{ellinia}{\hyphenchar\font45}
\DeclareFontShape{HE8}{ellinia}{m}{n}   {<-> [\Elliniascale] ellinia}{}
\DeclareFontShape{HE8}{ellinia}{m}{it}  {<-> [\Elliniascale] elliniai}{}
\DeclareFontShape{HE8}{ellinia}{m}{sl}  {<-> [\Elliniascale] elliniai}{}
\DeclareFontShape{HE8}{ellinia}{b}{n}   {<-> [\Elliniascale] elliniab}{}
\DeclareFontShape{HE8}{ellinia}{bx}{n}  {<-> [\Elliniascale] elliniab}{}
\DeclareFontShape{HE8}{ellinia}{bx}{it} {<-> [\Elliniascale] elliniabi}{}
%\endinput % is it needed [tzafrir]
⟨/HE8ellinia⟩
```

### 62.6.9 8Bit FrankRuehl font

*FrankRuehl* is a serif hebrew font created by the omega project [FILL IN CREDITS] [FILL IN GENERAL SHAPE DESCRIPTION] shapes: [FILL IN]

```
⟨*HE8frankruehl⟩
\def\FrankRuehlscale{1.0}
\DeclareFontFamily{HE8}{frank}{\hyphenchar\font45}
\DeclareFontShape{HE8}{frank}{m}{n}   {<-> [\FrankRuehlscale] frank}{}
\DeclareFontShape{HE8}{frank}{m}{it}  {<-> [\FrankRuehlscale] franki}{}
\DeclareFontShape{HE8}{frank}{m}{sl}  {<-> [\FrankRuehlscale] franki}{}
\DeclareFontShape{HE8}{frank}{b}{n}   {<-> [\FrankRuehlscale] frankb}{}
\DeclareFontShape{HE8}{frank}{bx}{n}  {<-> [\FrankRuehlscale] frankb}{}
\DeclareFontShape{HE8}{frank}{bx}{it} {<-> [\FrankRuehlscale] frankbi}{}
%\endinput % is it needed [tzafrir]
⟨/HE8frankruehl⟩
```

### 62.6.10 8Bit KtavYad font

*KtavYad* is a serif hebrew font created by the omega project [FILL IN CREDITS] [FILL IN GENERAL SHAPE DESCRIPTION] shapes: [FILL IN]

```
⟨*HE8yad⟩
\def\KtavYadscale{1.0}
\DeclareFontFamily{HE8}{yad}{\hyphenchar\font45}
\DeclareFontShape{HE8}{yad}{m}{n} {<-> [\KtavYadscale] yadi}{}
\DeclareFontShape{HE8}{yad}{m}{it} {<-> [\KtavYadscale] yadi}{}
\DeclareFontShape{HE8}{yad}{m}{sl} {<-> [\KtavYadscale] yadi}{}
\DeclareFontShape{HE8}{yad}{b}{n} {<-> [\KtavYadscale] yadbi}{}
\DeclareFontShape{HE8}{yad}{bx}{n} {<-> [\KtavYadscale] yadbi}{}
```

```
\DeclareFontShape{HE8}{yad}{bx}{it} {<-> [\KtavYadscale] yadbi}{}
%\endinput % is it needed [tzafrir]
⟨/HE8yad⟩
```

### 62.6.11 8Bit MiriamMono font

*MiriamMono* is a serif hebrew font created by the omega project [FILL IN CREDITS] [FILL IN GENERAL SHAPE DESCRIPTION] shapes: [FILL IN]

```
⟨*HE8miriam⟩
\def\MiriamMonoscale{1.0}
\DeclareFontFamily{HE8}{miriam}{\hyphenchar\font45}
\DeclareFontShape{HE8}{miriam}{m}{n}   {<-> [\MiriamMonoscale] miriam}{}
\DeclareFontShape{HE8}{miriam}{m}{it}  {<-> [\MiriamMonoscale] miriami}{}
\DeclareFontShape{HE8}{miriam}{m}{sl}  {<-> [\MiriamMonoscale] miriami}{}
\DeclareFontShape{HE8}{miriam}{b}{n}   {<-> [\MiriamMonoscale] miriamb}{}
\DeclareFontShape{HE8}{miriam}{bx}{n}  {<-> [\MiriamMonoscale] miriamb}{}
\DeclareFontShape{HE8}{miriam}{bx}{it} {<-> [\MiriamMonoscale] miriambi}{}

%\endinput % is it needed [tzafrir]
⟨/HE8miriam⟩
```

### 62.6.12 8Bit Nachlieli font

*Nachlieli* is a serif hebrew font created by the omega project [FILL IN CREDITS] [FILL IN GENERAL SHAPE DESCRIPTION] shapes: [FILL IN]

```
⟨*HE8nachlieli⟩
\def\Nachlieliscale{1.0}
\DeclareFontFamily{HE8}{nachlieli}{\hyphenchar\font45}
\DeclareFontShape{HE8}{nachlieli}{m}{n}   {<-> [\Nachlieliscale] nachlieli}{}
\DeclareFontShape{HE8}{nachlieli}{m}{it}  {<-> [\Nachlieliscale] nachlielii}{}
\DeclareFontShape{HE8}{nachlieli}{m}{sl}  {<-> [\Nachlieliscale] nachlielii}{}
\DeclareFontShape{HE8}{nachlieli}{b}{n}   {<-> [\Nachlieliscale] nachlielib}{}
\DeclareFontShape{HE8}{nachlieli}{bx}{n}  {<-> [\Nachlieliscale] nachlielib}{}
\DeclareFontShape{HE8}{nachlieli}{bx}{it} {<-> [\Nachlieliscale] nachlielibi}{}
%\endinput % is it needed [tzafrir]
⟨/HE8nachlieli⟩
```

### 62.6.13 Hebrew font switching commands

The `hebfont` package defines a number of high-level commands (all starting with `\text..` similar to the standard LaTeX 2ε font-change commands, for example `\textbf`) that have one argument and typeset this argument in the requested way. These commands are defined for all available Hebrew fonts defined above and change only font parameters but not direction.

For example, to use Hebrew Classic font family, the following sequence of commands should be included in a LaTeX 2ε document:

```
\sethebrew
\textclas{Hebrew text printed with Classic fonts}
```

| *Command* | *Corresponds to* | *Font family* |
|---|---|---|
| `\textjm{..}` | `\rmfamily` | Jerusalem font |
| `\textds{..}` | `\bfseries` | Dead Sea font |
| `\textoj{..}` | `\itshape` `\slshape` `\emph` | Old Jaffa font |
| `\textta{..}` | `\sffamily` `\ttfamily` | Tel-Aviv font |
| `\textcrml{..}` | `\fontfamily{crml}` | Carmel fonts |
| `\textfr{..}` | `\fontfamily{fr}` | Frank-Ruehl fonts |
| `\textredis{..}` | `\fontfamily{redis}` | Redis fonts |
| `\textclas{..}` | `\fontfamily{redis}` | Classic fonts |
| `\textshold{..}` | `\fontfamily{shold}` | Shalom Old Style font |
| `\textshscr{..}` | `\fontfamily{shscr}` | Shalom Script font |
| `\textshstk{..}` | `\fontfamily{shstk}` | Shalom Stick font |

Table 33: Hebrew font-change commands with arguments

The font change commands provided here all start with `\text..` to emphasize that they are for use in normal text and to be easily memorable.

or to use Hebrew with Classic fonts locally:

```
\R{\textclas{Hebrew text printed with Classic fonts}}
```

We declare LaTeX $2_\varepsilon$ font commands, e.g. `\textjm{...}` for all available fonts. Table 33 shows the meanings of all these new high-level commands.

`\textjm` Switches to *Jerusalem* font which is default regular Hebrew font ("roman" family). Commands `\textrm{...}` and old-style `{\rm ...}` will produce the same result.

```
⟨*hebfont⟩
\def\ivritex@tmp{HE8}
\ifx\ivritex@tmp\HeblatexEncoding %
  % compatibility with hebfonts:
  \DeclareTextFontCommand{\textjm}{\rmfamily\selectfont}
  \DeclareTextFontCommand{\textds}{\bfseries\selectfont}
  \DeclareTextFontCommand{\textoj}{\itshape\selectfont}
  \DeclareTextFontCommand{\textta}{\sffamily\selectfont}

  % an attempt to give some replacements to the original hebfonts:
  %
  \DeclareTextFontCommand{\textcrml}{\fontfamily{david}\selectfont}
  \DeclareTextFontCommand{\textfr}{\fontfamily{frank}\selectfont}
  \DeclareTextFontCommand{\textredis}{\fontfamily{aharoni}\selectfont}
  \DeclareTextFontCommand{\textclas}{\fontfamily{drugulin}\selectfont}
  \DeclareTextFontCommand{\textshold}{\fontfamily{frank}\selectfont}
  \DeclareTextFontCommand{\textshscr}{\fontfamily{yad}\selectfont}
  \DeclareTextFontCommand{\textshstk}{\fontfamily{aharoni}\selectfont}
  % note that redis is larger than shstk


  \DeclareTextFontCommand{\textaha}{\fontfamily{aharoni}\selectfont}
  \DeclareTextFontCommand{\textdav}{\fontfamily{david}\selectfont}
  \DeclareTextFontCommand{\textdru}{\fontfamily{drugulin}\selectfont}
  \DeclareTextFontCommand{\textel} {\fontfamily{ellinia}\selectfont}
  % \textfr is already declared above
  \DeclareTextFontCommand{\textmir}{\fontfamily{miriam}\selectfont}
  \DeclareTextFontCommand{\textna} {\fontfamily{nachlieli}\selectfont}
  % is this necessary:
  \DeclareTextFontCommand{\textyad} {\fontfamily{yad}\selectfont}

\else%
\DeclareTextFontCommand{\textjm}{\rmfamily\selectfont}
```

`\textds` Switches to *Dead Sea* font which is default bold font in Hebrew. Commands `\textbf{...}` and old-style `{\bf ...}` will produce the same result.

```
\DeclareTextFontCommand{\textds}{\bfseries\selectfont}
```

`\textoj` Switches to *Old Jaffa* font which is default italic font in Hebrew. Commands `\textit{...}`, `\textsl{...}`, `\emph{...}` and old-style `{\it ...}` or `{\em ...}` will produce the same result.

```
\DeclareTextFontCommand{\textoj}{\itshape\selectfont}
```

`\textta` Switches to *Tel-Aviv* font which is default sans-serif font in Hebrew. Commands `\textsf{...}`, `\texttt{...}` and old-style `{\sf ...}` or `{\tt ...}` will produce the same result (because sans-serif is used as typewriter font when in Hebrew mode).

```
\DeclareTextFontCommand{\textta}{\sffamily\selectfont}
```

`\textcrml` Switches to *Carmel* font. Regular and slanted variants of carmel font will be used..

```
\DeclareTextFontCommand{\textcrml}{\fontfamily{crml}\selectfont}
```

`\textfr` Switches to *Frank-Ruehl* font family. Regular, bold and slanted frank ruehl fonts will be used.

```
\DeclareTextFontCommand{\textfr}{\fontfamily{fr}\selectfont}
```

`\textredis` Switches to *Redis* font family. Regular, bold and slanted redis fonts of various sizes will be used.

```
\DeclareTextFontCommand{\textredis}{\fontfamily{redis}\selectfont}
```

`\textclas` Switches to *Classic* font family. The normal font will be hclassic and slanted — hcaption.

```
\DeclareTextFontCommand{\textclas}{\fontfamily{clas}\selectfont}
```

| *Old font command* | *Font name* | *Comment* |
|---|---|---|
| `{\jm ..}` | Jerusalem | default regular (roman) font |
| `{\ds ..}` | Dead Sea | default bold font |
| `{\oj ..}` | Old Jaffa | default italic and slanted font used also to emphasize text |
| `{\ta ..}` | Tel-Aviv | default sans-serif and typewriter font |

Table 34: Hebrew old font-change commands for compatibility mode

`\textshold` Switches to *Shalom Old Style* font.

```
\DeclareTextFontCommand{\textshold}{\fontfamily{shold}\selectfont}
```

`\textshscr` Switches to *Shalom Script* font.

```
\DeclareTextFontCommand{\textshscr}{\fontfamily{shscr}\selectfont}
```

`\textshstk` Switches to *Shalom Stick* font.

```
\DeclareTextFontCommand{\textshstk}{\fontfamily{shstk}\selectfont}
\fi
```

Finally, for backward compatibility with LaTeX2.09. four old font commands, e.g. `{\jm ...}` are defined too (see Table 34).

```
\if@compatibility
  \DeclareOldFontCommand{\jm}{\normalfont\rmfamily\selectfont}%
                             {\@nomath\jm}
  \DeclareOldFontCommand{\ds}{\normalfont\bfseries\selectfont}%
                             {\@nomath\ds}
  \DeclareOldFontCommand{\oj}{\normalfont\itshape\selectfont}%
                             {\@nomath\oj}
  \DeclareOldFontCommand{\ta}{\normalfont\sffamily\selectfont}%
                             {\@nomath\ta}
\fi
⟨/hebfont⟩
```

## 63 Hebrew in LaTeX 2.09 compatibility mode

`\documentstyle` command in the preamble of LaTeX document indicates that it is a LaTeX 2.09 document, and should be processed in *compatibility mode.* In such documents, one of the following three Hebrew style options can be included:

1. `hebrew_newcode` indicates that document will use UNIX ISO 8859-8 or Windows cp1255 input encoding, i.e. *Alef* letter will be represented as 224.

2. `hebrew_p` indicates that document is encoded with IBM PC cp862 encoding, i.e. *Alef* letter will be represented as 128.

3. `hebrew_oldcode` indicates that document uses old 7-bit encoding, as defined in Israeli Standard 960, i.e. *Alef* is character number 96.

Note, that other hebrew-related styles, such as `hebcal` can be included *after* the abovenamed Hebrew style option, for example:

`\documentstyle[12pt,hebrew_p,hebcal]{report}`.

Any Hebrew document which compiled under LaTeX 2.09 should compile under compatibility mode, unless it uses low-level commands such as `\tenrm`.

## 63.1 The DOCSTRIP modules

The following modules are used in the implementation to direct DOCSTRIP in generating the external files:

| | |
|---|---|
| newcode | produce `hebrew_newcode.sty` |
| pccode | produce `hebrew_p.sty` |
| oldcode | produce `hebrew_oldcode.sty` |

## 63.2 Obsolete style files

For each of the Hebrew LaTeX 2.09 Hebrew styles, we produce a file which uses correct input encoding and calls babel with Hebrew and English language options. This means that any styles which say `\input hebrew_newcode.sty` or `\documentstyle[...hebrew_newcode...]{...}` should still work.

```
⟨*newcode | pccode | oldcode⟩
\NeedsTeXFormat{LaTeX2e}
⟨/newcode | pccode | oldcode⟩

⟨*newcode⟩
\@obsoletefile{hebrew.sty}{hebrew_newcode.sty}
\RequirePackage[8859-8]{inputenc}
⟨/newcode⟩
⟨*pccode⟩
\@obsoletefile{hebrew.sty}{hebrew_p.sty}
\RequirePackage[cp862]{inputenc}
⟨/pccode⟩
⟨*oldcode⟩
\@obsoletefile{hebrew.sty}{hebrew_oldcode.sty}
\RequirePackage[si960]{inputenc}
⟨/oldcode⟩

⟨*newcode | pccode | oldcode⟩
\RequirePackage[english,hebrew]{babel}
⟨/newcode | pccode | oldcode⟩
```

## 64 The Bahasa language

The file `bahasa.dtx`[74] defines all the language definition macros for the bahasa indonesia / bahasa melayu language. Bahasa just means ‘language’ in bahasa indonesia / bahasa melayu. Since both national versions of the language use the same writing, although differing in pronounciation, this file can be used for both languages.

For this language currently no special definitions are needed or available.

The macro `\LdfInit` takes care of preventing that this file is loaded more than once, checking the category code of the `@` sign, etc.

```
⟨*code⟩
\LdfInit{bahasa}\captionsbahasa
```

When this file is read as an option, i.e. by the `\usepackage` command, `bahasa` could be an ‘unknown’ language in which case we have to make it known. So we check for the existence of `\l@bahasa` to see whether we have to do something here.

```
\ifx\l@bahasa\@undefined
  \@nopatterns{Bahasa}
  \adddialect\l@bahasa0\fi
```

The next step consists of defining commands to switch to (and from) the Bahasa language.

`\captionsbahasa` The macro `\captionsbahasa` defines all strings used in the four standard documentclasses provided with LaTeX.

```
\addto\captionsbahasa{%
  \def\prefacename{Pendahuluan}%
  \def\refname{Pustaka}%
  \def\abstractname{Ringkasan}% (sometime it's called 'intisari'
                               %  or 'ikhtisar')
  \def\bibname{Bibliografi}%
  \def\chaptername{Bab}%
  \def\appendixname{Lampiran}%
  \def\contentsname{Daftar Isi}%
  \def\listfigurename{Daftar Gambar}%
  \def\listtablename{Daftar Tabel}%
  \def\indexname{Indeks}%
  \def\figurename{Gambar}%
  \def\tablename{Tabel}%
  \def\partname{Bagian}%
%  Subject:  Subyek
%  From:  Dari
  \def\enclname{Lampiran}%
  \def\ccname{cc}%
  \def\headtoname{Kepada}%
  \def\pagename{Halaman}%
%  Notes (Endnotes): Catatan
```

[74]The file described in this section has version number v1.0i and was last revised on 2005/03/29.

```
  \def\seename{lihat}%
  \def\alsoname{lihat juga}%
  \def\proofname{Bukti}%
  \def\glossaryname{Daftar Istilah}%
  }
```

\datebahasa The macro `\datebahasa` redefines the command `\today` to produce Bahasa dates.

```
\def\datebahasa{%
  \def\today{\number\day~\ifcase\month\or
    Januari\or Pebruari\or Maret\or April\or Mei\or Juni\or
    Juli\or Agustus\or September\or Oktober\or Nopember\or Desember\fi
    \space \number\year}}
```

\extrasbahasa \noextrasbahasa The macro `\extrasbahasa` will perform all the extra definitions needed for the Bahasa language. The macro `\extrasbahasa` is used to cancel the actions of `\extrasbahasa`. For the moment these macros are empty but they are defined for compatibility with the other language definition files.

```
\addto\extrasbahasa{}
\addto\noextrasbahasa{}
```

\bahasahyphenmins The bahasa hyphenation patterns should be used with `\lefthyphenmin` set to 2 and `\righthyphenmin` set to 2.

```
\providehyphenmins{\CurrentOption}{\tw@\tw@}
```

The macro `\ldf@finish` takes care of looking for a configuration file, setting the main language to be switched on at `\begin{document}` and resetting the category code of `@` to its original value.

```
\ldf@finish{bahasa}
⟨/code⟩
```

## 65 Not renaming hyphen.tex

As Don Knuth has declared that the filename **hyphen.tex** may only be used to designate *his* version of the american English hyphenation patterns, a new solution has to be found in order to be able to load hyphenation patterns for other languages in a plain-based TEX-format. When asked he responded:

> That file name is "sacred", and if anybody changes it they will cause severe upward/downward compatibility headaches.
>
> People can have a file localhyphen.tex or whatever they like, but they mustn't diddle with hyphen.tex (or plain.tex except to preload additional fonts).

The files `bplain.tex` and `blplain.tex` can be used as replacement wrappers around `plain.tex` and `lplain.tex` to acheive the desired effect, based on the `babel` package. If you load each of them with iniTEX, you will get a file called either `bplain.fmt` or `blplain.fmt`, which you can use as replacements for `plain.fmt` and `lplain.fmt`.

As these files are going to be read as the first thing iniTEX sees, we need to set some category codes just to be able to change the definition of `\input`

```
⟨*bplain | blplain⟩
\catcode`\{=1 % left brace is begin-group character
\catcode`\}=2 % right brace is end-group character
\catcode`\#=6 % hash mark is macro parameter character
```

Now let's see if a file called `hyphen.cfg` can be found somewhere on TEX's input path by trying to open it for reading...

```
\openin 0 hyphen.cfg
```

If the file wasn't found the following test turns out true.

```
\ifeof0
\else
```

When `hyphen.cfg` could be opened we make sure that *it* will be read instead of the file `hyphen.tex` which should (according to Don Knuth's ruling) contain the american English hyphenation patterns and nothing else.

We do this by first saving the original meaning of `\input` (and I use a one letter control sequence for that so as not to waste multi-letter control sequence on this in the format).

```
  \let\a\input
```

Then `\input` is defined to forget about its argument and load `hyphen.cfg` instead.

```
  \def\input #1 {%
    \let\input\a
    \a hyphen.cfg
```

Once that's done the original meaning of `\input` can be restored and the definition of `\a` can be forgotten.

```
    \let\a\undefined
  }
```

```
\fi
⟨/bplain | blplain⟩
```

Now that we have made sure that `hyphen.cfg` will be loaded at the right moment it is time to load `plain.tex`.

```
⟨bplain⟩\a plain.tex
⟨blplain⟩\a lplain.tex
```

Finally we change the contents of `\fmtname` to indicate that this is *not* the plain format, but a format based on plain with the `babel` package preloaded.

```
⟨bplain⟩\def\fmtname{babel-plain}
⟨blplain⟩\def\fmtname{babel-lplain}
```

When you are using a different format, based on plain.tex you can make a copy of blplain.tex, rename it and replace `plain.tex` with the name of your format file.

## 66 Support for formats based on PLAINTEX

The following code duplicates or emulates parts of LaTeX 2$_\varepsilon$ that are needed for babel.

```
⟨*code⟩
\ifx\adddialect\@undefined
```

When `\adddialect` is still undefined we are making a format. In that case only the first part of this file is needed.

```
  \def\@empty{}
```

We need to define `\loadlocalcfg` for plain users as the LaTeX definition uses `\InputIfFileExists`.

```
  \def\loadlocalcfg#1{%
    \openin0#1.cfg
    \ifeof0
      \closein0
    \else
      \closein0
      {\immediate\write16{************************************}%
       \immediate\write16{* Local config file #1.cfg used}%
       \immediate\write16{*}%
       }
      \input #1.cfg\relax
    \fi
```

We have to execute `\@endofldf` in this case

```
    \@endofldf
    }
```

We want to add a message to the message LaTeX 2.09 puts in the `\everyjob` register. This could be done by the following code:

```
\let\orgeveryjob\everyjob
\def\everyjob#1{%
  \orgeveryjob{#1}%
  \orgeveryjob\expandafter{\the\orgeveryjob\immediate\write16{%
      hyphenation patterns for \the\loaded@patterns loaded.}}%
  \let\everyjob\orgeveryjob\let\orgeveryjob\@undefined}
```

The code above redefines the control sequence `\everyjob` in order to be able to add something to the current contents of the register. This is necessary because the processing of hyphenation patterns happens long before LaTeX fills the register.

There are some problems with this approach though.

- When someone wants to use several hyphenation patterns with SLiTeX the above scheme won't work. The reason is that SLiTeX overwrites the contents of the `\everyjob` register with its own message.
- Plain TeX does not use the `\everyjob` register so the message would not be displayed.

To circumvent this a 'dirty trick' can be used. As this code is only processed when creating a new format file there is one command that is sure to be used, `\dump`. Therefore the original `\dump` is saved in `\org@dump` and a new definition is supplied.

```
\let\orig@dump=\dump
\def\dump{%
```

To make sure that LaTeX 2.09 executes the `\@begindocumenthook` we would want to alter `\begin{document}`, but as this done too often already, we add the new code at the front of `\@preamblecmds`. But we can only do that after it has been defined, so we add this piece of code to `\dump`.

```
  \ifx\@ztryfc\@undefined
  \else
    \toks0=\expandafter{\@preamblecmds}
    \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}
    \def\@begindocumenthook{}
  \fi
```

This new definition starts by adding an instruction to write a message on the terminal and in the transcript file to inform the user of the preloaded hyphenation patterns.

```
  \everyjob\expandafter{\the\everyjob%
    \immediate\write16{\the\toks8 loaded.}}%
```

Then everything is restored to the old situation and the format is dumped.

```
  \let\dump\orig@dump\let\orig@dump\@undefined\dump}
\expandafter\endinput
\fi
```

The rest of this file is not processed by iniTeX but during the normal document run. A number of LaTeX macro's that are needed later on.

```
\long\def\@firstofone#1{#1}
\long\def\@firstoftwo#1#2{#1}
```

```
\long\def\@secondoftwo#1#2{#2}
\def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
\def\@star@or@long#1{%
  \@ifstar
  {\let\l@ngrel@x\relax#1}%
  {\let\l@ngrel@x\long#1}}
\let\l@ngrel@x\relax
\def\@car#1#2\@nil{#1}
\def\@cdr#1#2\@nil{#2}
\let\@typeset@protect\relax
\long\def\@gobble#1{}
\edef\@backslashchar{\expandafter\@gobble\string\\}
\def\strip@prefix#1>{}
\def\g@addto@macro#1#2{{%
    \toks@\expandafter{#1#2}%
    \xdef#1{\the\toks@}}}
\def\@namedef#1{\expandafter\def\csname #1\endcsname}
\def\@ifundefined#1{%
  \expandafter\ifx\csname#1\endcsname\relax
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi}
```

LaTeX $2_{\varepsilon}$ has the command `\@onlypreamble` which adds commands to a list of commands that are no longer needed after `\begin{document}`.

```
\ifx\@preamblecmds\@undefined
  \def\@preamblecmds{}
\fi
\def\@onlypreamble#1{%
  \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
    \@preamblecmds\do#1}}
\@onlypreamble\@onlypreamble
```

Mimick LaTeX's `\AtBeginDocument`; for this to work the user needs to add `\begindocument` to his file.

```
\def\begindocument{%
  \@begindocumenthook
  \global\let\@begindocumenthook\@undefined
  \def\do##1{\global\let ##1\@undefined}%
  \@preamblecmds
  \global\let\do\noexpand
  }

\ifx\@begindocumenthook\@undefined
  \def\@begindocumenthook{}
\fi
\@onlypreamble\@begindocumenthook
\def\AtBeginDocument{\g@addto@macro\@begindocumenthook}
```

We also have to mimick LaTeX's `\AtEndOfPackage`. Our replacement macro is much simpler; it stores its argument in `\@endofldf`.

```
\def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
\@onlypreamble\AtEndOfPackage
\def\@endofldf{}
\@onlypreamble\@endofldf
```

LaTeX needs to be able to switch off writing to its auxiliary files; plain doesn't have them by default.

```
\ifx\if@filesw\@undefined
  \expandafter\let\csname if@filesw\expandafter\endcsname
    \csname iffalse\endcsname
\fi
```

Mimick LaTeX's commands to define control sequences.

```
\def\newcommand{\@star@or@long\new@command}
\def\new@command#1{%
  \@testopt{\@newcommand#1}0}
\def\@newcommand#1[#2]{%
  \@ifnextchar [{\@xargdef#1[#2]}%
                {\@argdef#1[#2]}}
\long\def\@argdef#1[#2]#3{%
  \@yargdef#1\@ne{#2}{#3}}
\long\def\@xargdef#1[#2][#3]#4{%
  \expandafter\def\expandafter#1\expandafter{%
    \expandafter\@protected@testopt\expandafter #1%
    \csname\string#1\expandafter\endcsname{#3}}%
  \expandafter\@yargdef \csname\string#1\endcsname
  \tw@{#2}{#4}}
\long\def\@yargdef#1#2#3{%
  \@tempcnta#3\relax
  \advance \@tempcnta \@ne
  \let\@hash@\relax
  \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
  \@tempcntb #2%
  \@whilenum\@tempcntb <\@tempcnta
  \do{%
    \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
    \advance\@tempcntb \@ne}%
  \let\@hash@##%
  \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
\let\providecommand\newcommand

\def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
\def\declare@robustcommand#1{%
   \edef\reserved@a{\string#1}%
   \def\reserved@b{#1}%
   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
   \edef#1{%
      \ifx\reserved@a\reserved@b
         \noexpand\x@protect
```

```
        \noexpand#1%
      \fi
      \noexpand\protect
      \expandafter\noexpand\csname
         \expandafter\@gobble\string#1 \endcsname
   }%
   \expandafter\new@command\csname
      \expandafter\@gobble\string#1 \endcsname
}
\def\x@protect#1{%
   \ifx\protect\@typeset@protect\else
      \@x@protect#1%
   \fi
}
\def\@x@protect#1\fi#2#3{%
   \fi\protect#1%
}
```

The following little macro `\in@` is taken from latex.ltx; it checks whether its first argument is part of its second argument. It uses the boolean `\in@`; allocating a new boolean inside conditionally executed code is not possible, hence the construct with the temporary definition of `\bbl@tempa`.

```
\def\bbl@tmpa{\csname newif\endcsname\ifin@}
\ifx\in@\@undefined
  \def\in@#1#2{%
    \def\in@@##1#1##2##3\in@@{%
      \ifx\in@##2\in@false\else\in@true\fi}%
    \in@@#2#1\in@\in@@}
\else
  \let\bbl@tmpa\@empty
\fi
\bbl@tmpa
```

LaTeX has a macro to check whether a certain package was loaded with specific options. The command has two extra arguments which are code to be executed in either the true or false case. This is used to detect whether the document needs one of the accents to be activated (activegrave and activeacute). For plain TeX we assume that the user wants them to be active by default. Therefore the only thing we do is execute the third argument (the code for the true case).

```
\def\@ifpackagewith#1#2#3#4{%
  #3}
```

The LaTeX macro `\@ifl@aded` checks whether a file was loaded. This functionality is not needed for plain TeX but we need the macro to be defined as a no-op.

```
\def\@ifl@aded#1#2#3#4{}
```

For the following code we need to make sure that the commands `\newcommand` and `\providecommand` exist with some sensible definition. They are not fully equivalent to their LaTeX $2_\varepsilon$ versions; just enough to make things work in plain TeXenvironments.

```
\ifx\@tempcnta\@undefined
  \csname newcount\endcsname\@tempcnta\relax
\fi
\ifx\@tempcntb\@undefined
  \csname newcount\endcsname\@tempcntb\relax
\fi
```

To prevent wasting two counters in LaTeX 2.09 (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (`\count10`).

```
\ifx\bye\@undefined
 \advance\count10 by -2\relax
\fi
\ifx\@ifnextchar\@undefined
  \def\@ifnextchar#1#2#3{%
    \let\reserved@d=#1%
    \def\reserved@a{#2}\def\reserved@b{#3}%
    \futurelet\@let@token\@ifnch}
  \def\@ifnch{%
    \ifx\@let@token\@sptoken
      \let\reserved@c\@xifnch
    \else
      \ifx\@let@token\reserved@d
        \let\reserved@c\reserved@a
      \else
        \let\reserved@c\reserved@b
      \fi
    \fi
    \reserved@c}
  \def\:{\let\@sptoken= } \:  % this makes \@sptoken a space token
  \def\:{\@xifnch} \expandafter\def\: {\futurelet\@let@token\@ifnch}
\fi
\def\@testopt#1#2{%
  \@ifnextchar[{#1}{#1[#2]}}
\def\@protected@testopt#1{%%
  \ifx\protect\@typeset@protect
    \expandafter\@testopt
  \else
    \@x@protect#1%
  \fi}
\long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
     #2\relax}\fi}
\long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
        \else\expandafter\@gobble\fi{#1}}
```

Code from `ltoutenc.dtx`, adapted for use in the plain TeX environment.

```
\def\DeclareTextCommand{%
   \@dec@text@cmd\providecommand
}
\def\ProvideTextCommand{%
```

```
   \@dec@text@cmd\providecommand
}
\def\DeclareTextSymbol#1#2#3{%
   \@dec@text@cmd\chardef#1{#2}#3\relax
}
\def\@dec@text@cmd#1#2#3{%
   \expandafter\def\expandafter#2%
      \expandafter{%
         \csname#3-cmd\expandafter\endcsname
         \expandafter#2%
         \csname#3\string#2\endcsname
      }%
%   \let\@ifdefinable\@rc@ifdefinable
   \expandafter#1\csname#3\string#2\endcsname
}
\def\@current@cmd#1{%
  \ifx\protect\@typeset@protect\else
      \noexpand#1\expandafter\@gobble
  \fi
}
\def\@changed@cmd#1#2{%
   \ifx\protect\@typeset@protect
      \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
         \expandafter\ifx\csname ?\string#1\endcsname\relax
            \expandafter\def\csname ?\string#1\endcsname{%
               \@changed@x@err{#1}%
            }%
         \fi
         \global\expandafter\let
           \csname\cf@encoding \string#1\expandafter\endcsname
           \csname ?\string#1\endcsname
      \fi
      \csname\cf@encoding\string#1%
        \expandafter\endcsname
   \else
      \noexpand#1%
   \fi
}
\def\@changed@x@err#1{%
    \errhelp{Your command will be ignored, type <return> to proceed}%
    \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
\def\DeclareTextCommandDefault#1{%
   \DeclareTextCommand#1?%
}
\def\ProvideTextCommandDefault#1{%
   \ProvideTextCommand#1?%
}
\expandafter\let\csname OT1-cmd\endcsname\@current@cmd
\expandafter\let\csname?-cmd\endcsname\@changed@cmd
\def\DeclareTextAccent#1#2#3{%
```

```
  \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
}
\def\DeclareTextCompositeCommand#1#2#3#4{%
   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
   \edef\reserved@b{\string##1}%
   \edef\reserved@c{%
     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
   \ifx\reserved@b\reserved@c
      \expandafter\expandafter\expandafter\ifx
         \expandafter\@car\reserved@a\relax\relax\@nil
         \@text@composite
      \else
         \edef\reserved@b##1{%
            \def\expandafter\noexpand
               \csname#2\string#1\endcsname####1{%
               \noexpand\@text@composite
                  \expandafter\noexpand\csname#2\string#1\endcsname
                  ####1\noexpand\@empty\noexpand\@text@composite
                  {##1}%
            }%
         }%
         \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
      \fi
      \expandafter\def\csname\expandafter\string\csname
         #2\endcsname\string#1-\string#3\endcsname{#4}
   \else
     \errhelp{Your command will be ignored, type <return> to proceed}%
     \errmessage{\string\DeclareTextCompositeCommand\space used on
         inappropriate command \protect#1}
   \fi
}
\def\@text@composite#1#2#3\@text@composite{%
   \expandafter\@text@composite@x
      \csname\string#1-\string#2\endcsname
}
\def\@text@composite@x#1#2{%
   \ifx#1\relax
      #2%
   \else
      #1%
   \fi
}
%
\def\@strip@args#1:#2-#3\@strip@args{#2}
\def\DeclareTextComposite#1#2#3#4{%
   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
   \bgroup
      \lccode`\@=#4%
      \lowercase{%
   \egroup
```

```
      \reserved@a @%
   }%
}
%
\def\UseTextSymbol#1#2{%
%   \let\@curr@enc\cf@encoding
%   \@use@text@encoding{#1}%
   #2%
%   \@use@text@encoding\@curr@enc
}
\def\UseTextAccent#1#2#3{%
%   \let\@curr@enc\cf@encoding
%   \@use@text@encoding{#1}%
%   #2{\@use@text@encoding\@curr@enc\selectfont#3}%
%   \@use@text@encoding\@curr@enc
}
\def\@use@text@encoding#1{%
%   \edef\f@encoding{#1}%
%   \xdef\font@name{%
%      \csname\curr@fontshape/\f@size\endcsname
%   }%
%   \pickup@font
%   \font@name
%   \@@enc@update
}
\def\DeclareTextSymbolDefault#1#2{%
   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
}
\def\DeclareTextAccentDefault#1#2{%
   \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
}
\def\cf@encoding{OT1}
```

Currently we only use the LaTeX $2_\varepsilon$ method for accents for those that are known to be made active in *some* language definition file.

```
\DeclareTextAccent{\"}{OT1}{127}
\DeclareTextAccent{\'}{OT1}{19}
\DeclareTextAccent{\^}{OT1}{94}
\DeclareTextAccent{\`}{OT1}{18}
\DeclareTextAccent{\~}{OT1}{126}
```

The following control sequences are used in `babel.def` but are not defined for PLAIN TeX.

```
\DeclareTextSymbol{\textquotedblleft}{OT1}{92}
\DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
\DeclareTextSymbol{\textquoteleft}{OT1}{`\`}
\DeclareTextSymbol{\textquoteright}{OT1}{`\'}
\DeclareTextSymbol{\i}{OT1}{16}
\DeclareTextSymbol{\ss}{OT1}{25}
```

For a couple of languages we need the LaTeX-control sequence `\scriptsize` to be available. Because plain TeX doesn't have such a sofisticated font mechanism as LaTeX has, we just `\let` it to `\sevenrm`.

```
\ifx\scriptsize\@undefined
  \let\scriptsize\sevenrm
\fi
⟨/code⟩
```

# Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.