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

Johannes Braams
Kooienswater 62
2715 AJ Zoetermeer
The Netherlands
J.L.Braams@cistron.nl

Printed July 11, 1995

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 multilingual documents can be written.

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 basially consists of only two commands. These commands can be used to select another language or to find out what the current language is.

`language` The environment `language` does basically the same as `\selectlanguage`, except the language change is local to the environment. For mixing left-to-right typesetting with right-to-left typesetting the use of this environment is a prerequisite.

`\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.

`\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.

`\languagename` The name of the current language is stored in the control sequence `\languagename`.

`\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` A command with one argument, the character to make active in order to define personal shorthands.

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

`\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.

2 Changes for LaTeX 2ε

With the advent of LaTeX 2ε the interface to babel in the preamble of the doument 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 LaTeX2e 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 3.5

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:

- 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 problaly 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`.

4 The interface between the core of babel and the language definition files

In the core of the babel system two macros are defined that are to be used 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 in the case where two languages can (or have to) use the same hyphenation patterns. This can be useful when a user wants to use a language for which no patterns are preloaded in the format. In such a case 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 have to conform to a number of conventions. The reason for this is that 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 such that they can be read by LATEX as well as by plain TEX. This 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*⟩. 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 languagedefinition 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.

\langhyphenmin 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 offer the user the possibility to switch between languages and we do not know in what state TEX might be 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*⟩.

\main@language To postpone the activation of the definitions needed for a language untill 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.

\loadlocalcfg At the end of the processing of 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*⟩ in order to support local document classes. The user will be informed of the fact that this configuration file is loaded.

4.1 Support for active characters

In quite a few 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 untill the end of then document. Its definition may vary.

\bbl@activate The command \bbl@activate is used to change the way an active character

\bbl@deactivate expands. \bbl@activate 'switches on' the active behaviour of the character. \bbl@deactive 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.i. ~ or "a and the code to be executed when the shorthand is encountered.

\bbl@add@special
\bbl@remove@special

"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.

4.2 Support for saving macro definitions

Language definition files may want to *re*define macros that already exist. Therefore 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*⟩.

The effect of the aforementioned macros is that a piece of code is appended 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.

4.3 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.

4.4 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 that 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.

[1] This mechanism was introduced by Bernd Raichle.

5 Compatibility with `german.sty`

As has been discussed before, 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.

6 Identification

The file `babel.sty`[3] is meant for LaTeX 2ε, therefore we make sure that the format file used is the right one.

```
6.1 ⟨+package⟩\NeedsTeXFormat{LaTeX2e}
```

The identification code for each file is something that was introduced in LaTeX 2ε. When the command `\ProvidesFile` does not exist, a dummy definition is provided.

```
6.2 ⟨*!package⟩
6.3 \ifx\ProvidesFile\undefined
6.4   \def\ProvidesFile#1[#2 #3 #4]{\wlog{#4 #3 <#2>}}\fi
6.5 ⟨/!package⟩
```

Identify each file that is produced from this source file.

```
6.6 ⟨+package⟩\ProvidesPackage{babel}
6.7 ⟨+core⟩\ProvidesFile{babel.def}
6.8 ⟨+kernel & patterns⟩\ProvidesFile{hyphen.cfg}
6.9 ⟨+kernel&!patterns⟩\ProvidesFile{switch.def}
6.10 ⟨+driver&!user⟩\ProvidesFile{babel.drv}
6.11 ⟨+driver & user⟩\ProvidesFile{user.drv}
6.12                    [1995/07/11 v3.5e
6.13 ⟨+package⟩       The Babel package]
6.14 ⟨+core⟩           Babel common definitions]
6.15 ⟨+kernel⟩        Babel language switching mechanism]
6.16 ⟨+driver⟩]
```

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

[3] The file described in this section is called `babel.dtx`, has version number ? and was last revised on ?.

7 The Package File

In order to make use of the new features of LaTeX 2$_\varepsilon$, a new file is introduced to the babel system, `babel.sty`. This file is loaded by the `\usepackage` command and defines all the language options known in the babelsystem.

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

'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 or the patterns to be loaded as 'english' 'american' or 'USenglish'.

```
⟨*package⟩
\DeclareOption{american}{%
  \ifx\l@USenglish\undefined
  \else
    \let\l@american\l@USenglish
  \fi
  \input{english.ldf}%
  \main@language{american}}
```

Austrian is really a dialect of German.

```
\DeclareOption{austrian}{%
  \input{germanb.ldf}%
  \main@language{austrian}}
\DeclareOption{bahasa}{\input{bahasa.ldf}}
\DeclareOption{brazil}{\input{portuges.ldf}\main@language{brazil}}

\DeclareOption{brazilian}{%
  \input{portuges.ldf}%
  \let\captionsbrazilian\captionsbrazil
  \let\datebrazilian\datebrazil
  \let\extrasbrazilian\extrasbrazil
  \let\noextrasbrazilian\noextrasbrazil
  \let\brazilianhyphenmins\brazilhyphenmins
  \main@language{brazilian}}
\DeclareOption{breton}{\input{breton.ldf}}

\DeclareOption{british}{%
  \ifx\l@british\undefined
    \ifx\l@UKenglish\undefined
    \else
      \let\l@english\l@UKenglish
    \fi
  \else
    \let\l@english\l@british
  \fi
  \input{english.ldf}%
  \ifx\l@british\undefined
    \let\l@british\l@english
  \fi
  \let\captionsbritish\captionsenglish
  \let\datebritish\dateenglish
  \let\extrasbritish\extrasenglish
  \let\britishhyphenmins\englishhyphenmins
  \main@language{british}
  }
```

```
\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}}
```

We allow for the british english patterns to be loaded as either 'english' or 'UKenglish'

```
\DeclareOption{english}{%
  \ifx\l@UKenglish\undefined
  \else
    \let\l@english\l@UKenglish
  \fi
  \input{english.ldf}%
  }
\DeclareOption{esperanto}{\input{esperant.ldf}}

\DeclareOption{estonian}{\input{estonian.ldf}}
\DeclareOption{finnish}{\input{finnish.ldf}}
```

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

```
\DeclareOption{francais}{%
  \ifx\l@francais\undefined
    \let\l@francais\l@french
  \fi
  \input{francais.ldf}%
  }
```

With LaTeX 2_ε we can now also use the option french and still call the file `francais.ldf`.

```
\DeclareOption{french}{%
  \ifx\l@french\undefined
    \let\l@french\l@francais
  \fi
  \IfFileExists{french.ldf}{%
    \input{french.ldf}}{%
    \input{francais.ldf}}%
  \let\captionsfrench\captionsfrancais
  \let\datefrench\datefrancais
  \let\extrasfrench\extrasfrancais
  \let\noextrasfrench\noextrasfrancais
  \let\frenchhyphenmins\francaishyphenmins
  }
\DeclareOption{galician}{\input{galician.ldf}}
\DeclareOption{german}{\input{germanb.ldf}}
\DeclareOption{germanb}{\input{germanb.ldf}}
```

hungarian is just a synonym for magyar

```
\DeclareOption{hungarian}{\input{magyar.ldf}%
  \let\captionshungarian\captionsmagyar
  \let\datehungarian\datemagyar
  \let\extrashungarian\extrasmagyar
  \let\noextrashungarian\noextrasmagyar
```

```
  \let\hugarianhyphenmins\magyarhyphenmins
  }
\DeclareOption{irish}{\input{irish.ldf}}
\DeclareOption{italian}{\input{italian.ldf}}
\DeclareOption{lowersorbian}{\input{lsorbian.ldf}}
\DeclareOption{magyar}{\input{magyar.ldf}}
\DeclareOption{norsk}{\input{norsk.ldf}}
```

For Norwegian two spelling variants are provided.

```
\DeclareOption{nynorsk}{%
  \input{norsk.ldf}%
  \main@language{nynorsk}}
\DeclareOption{polish}{\input{polish.ldf}}
\DeclareOption{portuges}{\input{portuges.ldf}}
\DeclareOption{portuguese}{\input{portuges.ldf}%
  \let\captionsportuguese\captionsportuges
  \let\dateportuguese\dateportuges
  \let\extrasportuguese\extrasportuges
  \let\noextrasportuguese\noextrasportuges
  \let\portuguesehyphenmins\portugeshyphenmins
}
\DeclareOption{romanian}{\input{romanian.ldf}}
%\DeclareOption{russian}{\input{russian.ldf}}
\DeclareOption{scottish}{\input{scottish.ldf}}
\DeclareOption{spanish}{\input{spanish.ldf}}
\DeclareOption{slovak}{\input{slovak.ldf}}
\DeclareOption{slovene}{\input{slovene.ldf}}
\DeclareOption{swedish}{\input{swedish.ldf}}
\DeclareOption{turkish}{\input{turkish.ldf}}
\DeclareOption{uppersorbian}{\input{usorbian.ldf}}
```

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

```
\ProcessOptions*
```

⟨/package⟩

8 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.

```
\ifx\AtBeginDocument\undefined
  \input plain.def\relax
\fi
```

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

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

To communicate to the language definition files that the core of the babel system has been loaded, the following control sequence is just `\let` equal to `\relax`.

```
\let\babel@core@loaded\relax
⟨/core⟩
```

8.1 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
```

[4]This is because different operating systems sometimes use *very* different filenaming conventions.

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,

```
\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. Therefore `\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#2#3{%
  \expandafter\ifx\csname l@#1\endcsname\relax
    \@nolanerr{#1}%
  \else
    \ifnum\csname l@#1\endcsname=\language #2%
    \else#3\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.

```
\def\selectlanguage#1{%
  \edef\languagename{%
    \ifnum\escapechar=\expandafter`\string#1\empty
     \else \string#1\empty\fi}
  \expandafter\protect\csname selectlanguage \expandafter\endcsname
  \expandafter{\languagename}}
```

Because the command \selectlanguage could be used in a moving argument it expands to \protect\selectlanguage␣. Therefore, 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
```

Remark: If the `\selectlanguage` command is written to a file, a possible control sequence argument gets totally expanded to the string without the leading escape character. In the normal case we have to deal with the fact that the argument of `\selectlanguage` is totally unexpanded at the moment. There is only one disadvantage in the current implementation: `\originalTeX` contains this unexpanded argument and therefore needs more memory for its macro definition.

```
\expandafter\def\csname selectlanguage \endcsname#1{%
  \select@language{#1}%
```

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

```
  \if@filesw
    \protected@write\@auxout{}{\string\select@language{#1}}%
    \addtocontents{toc}{\string\select@language{#1}}%
    \addtocontents{lof}{\string\select@language{#1}}%
    \addtocontents{lot}{\string\select@language{#1}}%
  \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
    \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}%
    \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 somewhart 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.

```
8.74    \babel@savevariable\lefthyphenmin
8.75    \babel@savevariable\righthyphenmin
8.76    \expandafter\ifx\csname #1hyphenmins\endcsname\relax
8.77      \lefthyphenmin\tw@\righthyphenmin\thr@@
8.78    \else
8.79      \expandafter\expandafter\expandafter\set@hyphenmins
8.80        \csname #1hyphenmins\endcsname
8.81    \fi
8.82  \fi}
```

`otherlanguage` The `otherlanguage` environment can be used as an alternative to using the `\selectlanguage` declarative command. When you are typesetting a document with 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.

```
8.83 \long\def\otherlanguage#1{%
8.84   \def\languagename{#1}%
8.85   \csname selectlanguage \endcsname{#1}%
8.86   \ignorespaces
8.87   }
```

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.

```
8.88 \long\def\endotherlanguage{%
8.89   \originalTeX
8.90   \global\@ignoretrue\ignorespaces
8.91   }
```

`\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`.

```
8.92 \def\foreignlanguage#1#2{%
8.93   \begingroup
8.94     \def\languagename{#1}%
8.95     \expandafter\ifx\csname l@#1\endcsname\relax
8.96       \@nolanerr{#1}%
8.97     \else
8.98       \language=\csname l@#1\endcsname\relax
8.99       \csname extras#1\endcsname

8.100      \expandafter\ifx\csname #1hyphenmins\endcsname\relax
8.101        \lefthyphenmin\tw@\righthyphenmin\thr@@
```

```
      \else
        \expandafter\expandafter\expandafter\set@hyphenmins
          \csname #1hyphenmins\endcsname
      \fi
    \fi
    #2
    \csname noextras#1\endcsname
    \endgroup
  }
```

`\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}
```
⟨/kernel⟩

`\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 ath the beginning of the document.

```
⟨*core⟩
\def\main@language#1{\def\bbl@main@language{#1}}
```

The default is to use English as the main language.

```
\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⟩
```

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\undefined\originalTeX\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\undefined\babel@beginsave\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.

When the format knows about `\PackageError` it must be LATEX 2_ε, 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\@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*{\@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⟩
```

`\patterns@loaded` It has been suggested to add a remark to LaTeX's `\everyjob` message, stating which hyphenation patterns have been loaded. This can be done by first collecting (in a token register) the names when processing the file `language.dat` and afterwards adding a string to the `\everyjob` message. The token register is initially empty.

```
\newtoks\patterns@loaded \global\patterns@loaded={}
```

`\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 wether the line starts with `=`.

`\bbl@eq@` To be able to do that we need an `=`, stored in a macro.

```
\def\bbl@eq@{=}
```

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/{%
  \def\bbl@tmp{#1}
  \ifx\bbl@tmp\bbl@eq@
    \process@synonym#2/
  \else
    \process@language#1#2/%
  \fi
  }
```

`\process@synonym` This macro takes care of the lines which start with an `=`.

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

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

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

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

```
    \global\patterns@loaded\expandafter{\the\patterns@loaded#1, }%
```

Some pattern files contain assignments to `\lefthyphenmin` and `\righthyphenmin`. TEX does not keep track of these assignments. Therefore 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
    \input #2\relax
    \ifnum\lefthyphenmin=\m@ne
      \lefthyphenmin\tw@
      \righthyphenmin\thr@@
    \fi
```

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@tmp{#3}
    \ifx\bbl@tmp\@empty
    \else
      \ifx\bbl@tmp\space
      \else
        \input #3\relax
      \fi
    \fi
```

Finally we store the settings of `\lefthyphenmin` and `\righthyphenmin`.

```
    \expandafter\edef\csname #1hyphenmins\endcsname{%
      \the\lefthyphenmin\the\righthyphenmin}}
```

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 controlsequence.

```
    \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@tmp\undefined
\let\bbl@eq@\undefined
\let\bbl@line\undefined
```

We want to add a message to the message LaTeX 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 orginal `\dump` is saved in `\org@dump` and a new definition is supplied.

```
\let\orig@dump=\dump
\def\dump{%
```

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{Hyphenation patterns for \the\patterns@loaded
      loaded.}}%
```

Then everything is restored to the old situation and the format is dumped.

```
  \let\dump\orig@dump\let\orig@dump\undefined\dump}
```

Here the code for iniTeX ends.

```
⟨/patterns⟩
⟨/kernel⟩
```

8.2 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}
```

8.3 Support for active characters

\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 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].

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

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.

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

If the character is already active we don't do anything.

```
8.253       \ifcat\noexpand##1\noexpand~\relax
8.254       \else
```

Otherwise we write a message in the transcript file,

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

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

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

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

```
8.257         \ifx##1'%
8.258           \let\pr@m@s\bbl@pr@m@s
8.259         \fi
```

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'.

```
8.260         \lccode`~=`##1%
8.261         \lowercase{%
```

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

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

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

Define the character to expand to

\active@prefix ⟨*char*⟩ \active@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}}%
```

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}%
```

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\csname @sh@\string##1@sel\endcsname
      {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.

```
\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}%
```

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\csname @sh@\string##1@sel\endcsname
              {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\csname @sh@\string##1@sel\endcsname
              {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}%
        \fi}%
    }\x
```

`\active@prefix` The comand `\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
    \bbl@afterfi\protect#1\@gobble
  \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 avail-

able. This setting of this switch should be checked in the first level expansion of `\active@char`⟨*char*⟩.

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

`\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 `\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\tmp{#3}%
  \ifx\tmp\empty
    \expandafter\let\csname @sh@\string#2@sel\endcsname\bbl@scndcs
  \else
    \expandafter\let\csname @sh@\string#2@sel\endcsname\bbl@firstcs
  \fi
  \@namedef{#1@sh@\string#2\string#3@}{#4}}
```

`\textormath` Some of the shorthands that will be declared by the language definition files have to be useable 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 no user group; use language group 'english' and have a system group called 'system'.

```
\def\user@group{}
\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{%
  \def\user@group{user}%
  \initiate@active@char{#1}%
  \bbl@activate{#1}}
```

`\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}}
```

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}}
```

`\bbl@pr@m@s`

One of the internal macros that are involved in substituting `\prime` for each right quote in mathmode is `\pr@m@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.

```
\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
```

```
8.396           \egroup
8.397         \fi
8.398       \fi}%
8.399     }
8.400 \x
8.401 ⟨/core | shorthands⟩
```

Normally the ~ is active and expands to `\penalty\@M\␣`. When it is written to the `.aux` file 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. To acheive that the character ~ first needs to have character code 12 (other).

```
8.402 ⟨*core⟩
8.403 \catcode`~12\relax
8.404 \initiate@active@char{~}
8.405 \declare@shorthand{system}{~}{\penalty\@M\ }
8.406 \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. Therefore we define two macros here to store the position of the character in these encodings.

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

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

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

8.4 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.

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

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

```
8.413 \newcount\babel@savecnt
8.414 \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.

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

```
\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
```

8.5 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{%
  \ifx#1\undefined
    \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
}
```

8.6 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\allowhyphens{\penalty\@M \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{{\ifhmode
    \edef\@SF{\spacefactor\the\spacefactor}\else
    \let\@SF\empty \fi \leavevmode #1\@SF}}
```

8.7 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`.

8.8 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, therefore 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 then `OT1` ot `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 then `OT1` ot `T1` is used this glyph can still be typeset.

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

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

[8] The file described in this section has version number ?, and was last revised on ?.

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

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

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

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

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

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

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

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

8.9 Letters

\ij The dutch language uses the letter 'ij'. It is available in T1 encoded fonts, but not
\IJ in the OT1 encoded fonts. Therefore 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 then 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 availabel 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 my by Stipcevic Mario, (stipcevic@olimp.irb.hr).

```
\def\crrtic@{\hrule height0.1ex width0.3em}
\def\crttic@{\hrule height0.1ex width0.33em}
%
\def\dj@@#1#2#3{%
  \leavevmode\kern#1em\raise#2ex\vbox{\crrtic@}\kern#3em}
\def\dj@{%
  \ifcase\the\fam
  \dj@@{0.25}{1.28}{-0.50}\or %roman
  \dj@@{0.25}{1.28}{-0.50}\or %roman
  \dj@@{0.25}{1.28}{-0.50}\or %roman
  \dj@@{0.25}{1.28}{-0.50}\or %roman
  \dj@@{0.33}{1.29}{-0.63}\or %italic
  \dj@@{0.35}{1.30}{-0.65}\or %slanted
  \dj@@{0.25}{1.25}{-0.55}\or %bold
  \dj@@{0.25}{1.15}{-0.55}\or %tt
  \dj@@{-0.04}{0.72}{-0.26}\or%csc
  \dj@@{0.35}{1.30}{-0.65} %slanted
  \else\dj@@{0.25}{1.25}{-0.55}\fi}%bold
%
\def\DJ@@#1#2#3{%
  \leavevmode\kern#1em\raise#2ex\vbox{\crttic@}\kern#3em}
\def\DJ@{%
  \ifcase\the\fam
  \DJ@@{0.11}{0.92}{-0.36}\or
  \DJ@@{0.11}{0.92}{-0.36}\or
  \DJ@@{0.11}{0.92}{-0.36}\or
  \DJ@@{0.11}{0.92}{-0.36}\or
  \DJ@@{0.25}{0.92}{-0.50}\or
  \DJ@@{0.22}{0.92}{-0.47}\or
  \DJ@@{0.09}{0.90}{-0.34}\or
  \DJ@@{0.02}{0.85}{-0.27}\or
  \DJ@@{0.1}{0.86}{-0.35}\or
  \DJ@@{0.22}{0.92}{-0.47}\else
  \DJ@@{0.09}{0.90}{-0.34}\fi}
\DeclareTextCommand{\dj}{OT1}{\dj@ d}
\DeclareTextCommand{\DJ}{OT1}{\DJ@ D}
```

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

```
\ProvideTextCommandDefault{\dj}{%
```

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

8.10 Shorthands for quotation marks

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

`\glq` `\grq` The 'german' single quotes.

```
\DeclareRobustCommand{\glq}{%
  \ifmmode\mbox{\quotesinglbase}\else\quotesinglbase\fi}
\DeclareRobustCommand{\grq}{%
  \ifmmode\mbox{\textquoteleft}\else\textquoteleft\fi}
```

`\glqq` `\grqq` The 'german' double quotes.

```
\DeclareRobustCommand{\glqq}{%
  \ifmmode\mbox{\quotedblbase}\else\quotedblbase\fi}
\DeclareRobustCommand{\grqq}{%
  \ifmmode\mbox{\textquotedblleft}\else\textquotedblleft\fi}
```

`\flq` `\frq` The 'french' single quillemets.

```
\DeclareRobustCommand{\flq}{%
  \ifmmode\mbox{\quilsinglleft}\else\quilsinglleft\fi}
\DeclareRobustCommand{\frq}{%
  \ifmmode\mbox{\quilsinglright}\else\quilsinglright\fi}
```

`\flqq` `\frqq` The 'french' double quillemets.

```
\DeclareRobustCommand{\flqq}{%
  \ifmmode\mbox{\quillemotleft}\else\quillemotleft\fi}
\DeclareRobustCommand{\frqq}{%
  \ifmmode\mbox{\quillemotright}\else\quillemotright\fi}
```

8.11 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{{%
      \expandafter\accent\csname\f@encoding dqpos\endcsname
      ##1\allowhyphens}}%
  \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.

```
  {\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}}
```

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. Therefore 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}{O}{\bbl@umlauta{U}}%
}
```

8.12 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. Therefore 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%
```

8.12.1 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}}{}

```

[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`.

```
\@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}%
    \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⟩
```

8.13 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'.

`\babel@sanitize@arg` To call a macro with a 'sanitized' argument, instead of `\A{\B}` one would write `\babel@sanitize@arg{\A}{\B}`. (But be careful, this macro is not fully expandable!)

```
%tmp%\long\def\babel@sanitize@arg#1#2{\bgroup\def\@tempa{#2}%
%tmp%  \expandafter\babel@strip@meaning\meaning\@tempa\relax{#1}}
%tmp%\def\babel@strip@meaning#1->#2\relax#3{\egroup #3{#2}}
```

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 incase their definitions change (they have changed in the past).

`\label` The `\label` macro is one of the cross referencing macros affected. First we save its original definition.

```
%tmp%\let\LTX@label=\label
```

Then the macro `label` is redefined.

```
%tmp%\def\label#1{\babel@sanitize@arg\LTX@label{#1}}
```

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

```
⟨*core | shorthands⟩
\let\bbl@newlabel\newlabel
\def\newlabel#1#2{%
  \@safe@activestrue\bbl@newlabel{#1}{#2}\@safe@activesfalse}
```

`\@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.

```
\let\bbl@@testdef\@testdef
\def\@testdef#1#2#3{%
  \@safe@activestrue\bbl@@testdef{#1}{#2}{#3}\@safe@activesfalse}
```

`\ref` `\pageref` The same holds for the macro `\ref` that references a label and `\pageref` to reference a page. While we change these macros, we make them robust as well to prevent problems when they should become expanded at the wrong moment.

```
\let\bbl@ref\ref
\edef\ref{\noexpand\protect
```

```
  \expandafter\noexpand\csname bblref \endcsname}
\expandafter\def\csname bblref \endcsname#1{%
  \@safe@activestrue\bbl@ref{#1}\@safe@activesfalse}

\let\bbl@pageref\pageref
\edef\pageref{\noexpand\protect
  \expandafter\noexpand\csname bblpageref \endcsname}
\expandafter\def\csname bblpageref \endcsname#1{%
  \@safe@activestrue\bbl@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, so we redefine this internal macro and leave `\cite` alone.

```
%tmp%\let\bbl@@citex\@citex
%tmp%\def\@citex[#1]#2{\babel@sanitize@arg{\bbl@@citex[#1]}{#2}}
```

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

```
%tmp%\let\bbl@nocite\nocite
%tmp%\def\nocite#1{\babel@sanitize@arg\bbl@nocite{#1}}
```

`\bibcite` The macro that is used in the `.aux` file to define citation labels.

```
%tmp%\let\bbl@bibcite\bibcite
%tmp%\def\bibcite#1#2{\babel@sanitize@arg\bbl@bibcite{#1}{#2}}
```

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

```
%tmp%\let\bbl@@bibitem\@bibitem
%tmp%\def\@bibitem#1{\babel@sanitize@arg\bbl@@bibitem{#1}}
```

`\@lbibitem` The other of the two internal LATEX macros called by `\bibitem` that write the citation label on the `.aux` file.

```
%tmp%\let\bbl@@lbibitem\@lbibitem
%tmp%\def\@lbibitem[#1]#2{\babel@sanitize@arg{\bbl@@lbibitem[#1]}{#2}}
```

⟨/core | shorthands⟩

9 Local Language Configuration

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

```
⟨*core⟩
\def\loadlocalcfg#1{%
  \InputIfFileExists{#1.cfg}
            {\typeout{**************************************^^J%
                      * Local config file #1.cfg used^^J%
                      *}%
             }
            {}}
⟨/core⟩
```

10 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 in order 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 spcific files you can run the file `user.drv` through LaTeX to get a user guide.

```
10.1 ⟨*driver⟩
10.2 \documentclass{ltxdoc}
10.3 \DoNotIndex{\!,\',\,,\.,\-,\:,\;,\?,\/,\^,\‘,\@M}
10.4 \DoNotIndex{\@,\@ne,\@m,\@afterheading,\@date,\@endpart}
10.5 \DoNotIndex{\@hangfrom,\@idxitem,\@makeschapterhead,\@mkboth}
10.6 \DoNotIndex{\@oddfoot,\@oddhead,\@restonecolfalse,\@restonecoltrue}
10.7 \DoNotIndex{\@starttoc,\@unused}
10.8 \DoNotIndex{\accent,\active}
10.9 \DoNotIndex{\addcontentsline,\advance,\Alph,\arabic}
10.10 \DoNotIndex{\baselineskip,\begin,\begingroup,\bf,\box,\c@secnumdepth}
10.11 \DoNotIndex{\catcode,\centering,\char,\chardef,\clubpenalty}
10.12 \DoNotIndex{\columnsep,\columnseprule,\crcr,\csname}
10.13 \DoNotIndex{\day,\def,\dimen,\discretionary,\divide,\dp,\do}
10.14 \DoNotIndex{\edef,\else,\empty,\end,\endgroup,\endcsname,\endinput}
10.15 \DoNotIndex{\errhelp,\errmessage,\expandafter,\fi,\filedate}
10.16 \DoNotIndex{\fileversion,\fmtname,\fnum@figure,\fnum@table,\fontdimen}
10.17 \DoNotIndex{\gdef,\global}
10.18 \DoNotIndex{\hbox,\hidewidth,\hfil,\hskip,\hspace,\ht,\Huge,\huge}
10.19 \DoNotIndex{\ialign,\if@twocolumn,\ifcase,\ifcat,\ifhmode,\ifmmode}
10.20 \DoNotIndex{\ifnum,\ifx,\immediate,\ignorespaces,\input,\item}
10.21 \DoNotIndex{\kern}
10.22 \DoNotIndex{\labelsep,\Large,\large,\labelwidth,\lccode,\leftmargin}
10.23 \DoNotIndex{\lineskip,\leavevmode,\let,\list,\ll,\long,\lower}
10.24 \DoNotIndex{\m@ne,\mathchar,\mathaccent,\markboth,\month,\multiply}
10.25 \DoNotIndex{\newblock,\newbox,\newcount,\newdimen,\newif,\newwrite}
10.26 \DoNotIndex{\nobreak,\noexpand,\noindent,\null,\number}
10.27 \DoNotIndex{\onecolumn,\or}
10.28 \DoNotIndex{\p@,par, \parbox,\parindent,\parskip,\penalty}
10.29 \DoNotIndex{\protect,\ps@headings}
10.30 \DoNotIndex{\quotation}
10.31 \DoNotIndex{\raggedright,\raise,\refstepcounter,\relax,\rm,\setbox}
10.32 \DoNotIndex{\section,\setcounter,\settowidth,\scriptscriptstyle}
10.33 \DoNotIndex{\sfcode,\sl,\sloppy,\small,\space,\spacefactor,\strut}
10.34 \DoNotIndex{\string}
10.35 \DoNotIndex{\textwidth,\the,\thechapter,\thefigure,\thepage,\thepart}
10.36 \DoNotIndex{\thetable,\thispagestyle,\titlepage,\tracingmacros}
10.37 \DoNotIndex{\tw@,\twocolumn,\typeout,\uppercase,\usecounter}
10.38 \DoNotIndex{\vbox,\vfil,\vskip,\vspace,\vss}
10.39 \DoNotIndex{\widowpenalty,\write,\xdef,\year,\z@,\z@skip}
```

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

```
10.40 \makeatletter
10.41 \gdef\dlqq{{\setbox\tw@=\hbox{,}\setbox\z@=\hbox{''}%
10.42   \dimen\z@=\ht\z@ \advance\dimen\z@-\ht\tw@
10.43   \setbox\z@=\hbox{\lower\dimen\z@\box\z@}\ht\z@=\ht\tw@
10.44   \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*\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*\file[1]{\texttt{#1}}
\newcommand*\pkg[1]{\texttt{#1}}
\newcommand*\langdeffile[1]{%
⟨−user⟩  \clearpage
  \DocInput{#1}}
```

When a full index should be generated unomment the line with `\EnableCrossres`. 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{esperant.dtx}
\langdeffile{dutch.dtx}
\langdeffile{english.dtx}
\langdeffile{germanb.dtx}
%
\langdeffile{breton.dtx}
\langdeffile{irish.dtx}
\langdeffile{scottish.dtx}
%
\langdeffile{francais.dtx}
\langdeffile{italian.dtx}
\langdeffile{portuges.dtx}
\langdeffile{spanish.dtx}
\langdeffile{catalan.dtx}
\langdeffile{galician.dtx}
\langdeffile{romanian.dtx}
%
\langdeffile{danish.dtx}
\langdeffile{norsk.dtx}
\langdeffile{swedish.dtx}
%
\langdeffile{finnish.dtx}
\langdeffile{magyar.dtx}
\langdeffile{estonian.dtx}
%
\langdeffile{croatian.dtx}
\langdeffile{czech.dtx}
\langdeffile{polish.dtx}
\langdeffile{slovak.dtx}
\langdeffile{slovene.dtx}
%\langdeffile{russian.dtx}
%
\langdeffile{lsorbian.dtx}
\langdeffile{usorbian.dtx}
\langdeffile{turkish.dtx}
%
\langdeffile{bahasa.dtx}
```

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

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

10.127 ⟨/!user⟩
10.128 `\end{document}`
10.129 ⟨/driver⟩

11 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 to switch between languages in one document. The basic interface consists of using ones option, which is the same for *all* standard document classes.

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

12 Acknowledgements

I would like to thank all who volunteered as β-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. Werenfried Spit supplied the files for the Russian language. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polishing the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped finding and repairing 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 februari 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.

13 The Esperanto language

The file `esperant.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.

`^u`	gives ŭ, with hyphenation in the rest of the word allowed
`^U`	gives Ŭ, with hyphenation in the rest of the word allowed
`^h`	prevents ĥ from becoming too tall
`^j`	gives ĵ
`^|`	inserts a `\discretionary{-}{}{}`
`^a`	gives â with hyphenation in the rest of the word allowed, this works for all other letters as well.

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

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

```
13.1 ⟨*code⟩
13.2 \ifx\undefined\captionsesperanto
13.3 \else
13.4   \selectlanguage{esperanto}
13.5   \expandafter\endinput
13.6 \fi
```

`\atcatcode` This file, `esperant.ldf`, may have been read while TEX is in the middle of processing a document, so we have to make sure the category code of `@` is 'letter' while this file is being read. We save the category code of the @-sign in `\atcatcode` and make it 'letter'. Later the category code can be restored to whatever it was before.

```
13.7 \chardef\atcatcode=\catcode‘\@
13.8 \catcode‘\@=11\relax
```

Now we determine whether the the common macros from the file `babel.def` need to be read. We can be in one of two situations: either another language option has been read earlier on, in which case that other option has already read `babel.def`, or `esperanto` is the first language option to be processed. In that case we need to read `babel.def` right here before we continue.

```
13.9 \ifx\undefined\babel@core@loaded\input babel.def\relax\fi
```

Another check that has to be made, is if another language definition file has been read already. In that case its definitions have been activated. This might interfere with definitions this file tries to make. Therefore we make sure that we cancel any special definitions. This can be done by checking the existence of the

[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.

macro \originalTeX. If it exists we simply execute it, otherwise it is \let to \empty.

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

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\undefined\l@esperanto
  \@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}%
  \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{Proof}%  <-- needs translation
  }
```

\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 The macro \extrasesperanto performs all the extra definitions needed for the \noextrasesperanto 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{^}
\addto\extrasesperanto{\languageshorthands{esperanto}}
\addto\extrasesperanto{\bbl@activate{^}}
\addto\noextrasesperanto{\bbl@deactivate{^}}
```

And here are the uses of the active ^:

```
\declare@shorthand{esperanto}{^u}{\u u\allowhyphens}
\declare@shorthand{esperanto}{^U}{\u U\allowhyphens}
\declare@shorthand{esperanto}{^h}{h\llap{\^{}}\allowhyphens}
\declare@shorthand{esperanto}{^j}{\^{\j}\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 three steps. First the toplevel.

```
\def\esper#1{\@esper{\@nameuse{c@#1}}}
\def\Esper#1{\@Esper{\@nameuse{c@#1}}}
```

Then the first five occasions that are probably used the most.

```
\def\@esper#1{\ifcase#1\or a\or b\or c\or \^c\or d\else\@iesper{#1}\fi}
\def\@Esper#1{\ifcase#1\or A\or B\or C\or \^C\or D\else\@Iesper{#1}\fi}
```

And the 33 other cases.

```
\def\@iesper#1{\ifcase#1\or \or \or \or \or \or e\or f\or g\or \^g\or
    h\or h\llap{\^{}}\or i\or j\or \^\j\or k\orl\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\@Iesper#1{\ifcase#1\or \or \or \or \or \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}
```

It is possible that a site might need to add some extra code to the babel macros. To enable this we load a local configuration file, `esperant.cfg` if it is found on TEX' search path.

```
\loadlocalcfg{esperant}
```

Our last action is to make a note that the commands we have just defined, will be executed by calling the macro `\selectlanguage` at the beginning of the document.

```
\main@language{esperanto}
```

Finally, the category code of `@` is reset to its original value. The macrospace used by `\atcatcode` is freed.

13.68 `\catcode‘\@=\atcatcode \let\atcatcode\relax`

13.69 ⟨/code⟩

14 The Dutch language

The file `dutch.dtx`[11] defines all the language-specific macros for the Dutch language.

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.
`"'`	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`

As this file needs to be read only once, we check whether it was read before. If it was, the command `\captionsdutch` is already defined, so we can stop processing. If this command is undefined we proceed with the various definitions and first show the current version of this file.

```
14.1 ⟨*code⟩
14.2 \ifx\undefined\captionsdutch
14.3 \else
14.4   \selectlanguage{dutch}
14.5   \expandafter\endinput
14.6 \fi
```

`\atcatcode` This file, `dutch.ldf`, may have been read while TEX is in the middle of processing a document, so we have to make sure the category code of `@` is 'letter' while this file is being read. We save the category code of the @-sign in `\atcatcode` and make it 'letter'. Later the category code can be restored to whatever it was before.

```
14.7 \chardef\atcatcode=\catcode`\@
14.8 \catcode`\@=11\relax
```

Now we determine whether the common macros from the file `babel.def` need to be read. We can be in one of two situations: either another language option has

[11] The file described in this section has version number ?, and was last revised on ?.

been read earlier on, in which case that other option has already read `babel.def`, or `dutch` is the first language option to be processed. In that case we need to read `babel.def` right here before we continue.

```
\ifx\undefined\babel@core@loaded\input babel.def\relax\fi
```

Another check that has to be made, is if another language definition file has been read already. In that case its definitions have been activated. This might interfere with definitions this file tries to make. Therefore we make sure that we cancel any special definitions. This can be done by checking the existence of the macro `\originalTeX`. If it exists we simply execute it, otherwise it is `\let` to `\empty`.

```
\ifx\undefined\originalTeX \let\originalTeX\empty \fi
\originalTeX
```

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` to see whether we have to do something here.

```
\ifx\undefined\l@dutch
  \@nopatterns{Dutch}
  \adddialect\l@dutch0
\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
    \addto\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{Bewijs}%
      }
    }\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}}
```

\extrasdutch \noextrasdutch The macro \extrasdutch will perform all the extra definitions needed for the Dutch language. The macro \noextrasdutch is used to cancel the actions of \extrasdutch.

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{"}
\addto\extrasdutch{\languageshorthands{dutch}}
\addto\extrasdutch{\bbl@activate{"}}
%\addto\noextrasdutch{\bbl@deactivate{"}}
```

The 'umlaut' character should be positioned lower on *all* vowels in Dutch texts.

```
\addto\extrasdutch{\umlautlow\umlautelow}
\addto\noextrasdutch{\umlauthigh}
```

\dutchhyphenmins The dutch hyphenation patterns can be used with \lefthyphenmin set to 2 and \righthyphenmin set to 3.

```
\def\dutchhyphenmins{\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{\discretionary{-}{i}{\"{\i}}}{\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}{"-}{\allowhyphens-\allowhyphens}
\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}}
```

`\-` 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.

```
\addto\extrasdutch{\babel@save\-}
\addto\extrasdutch{\def\-{\allowhyphens
                          \discretionary{-}{}{}\allowhyphens}}
```

It is possible that a site might need to add some extra code to the babel macros. To enable this we load a local configuration file, `dutch.cfg` if it is found on TEX' search path.

```
\loadlocalcfg{dutch}
```

Our last action is to make a note that the commands we have just defined, will be executed by calling the macro `\selectlanguage` at the beginning of the document.

```
\main@language{dutch}
```

Finally, the category code of `@` is reset to its original value. The macrospace used by `\atcatcode` is freed.

```
\catcode`\@=\atcatcode \let\atcatcode\relax
```

⟨/code⟩

15 The English language

The file `english.dtx`[12] defines all the language definition macros for the English language as well as for the American version of this language.

For this language currently no special definitions are needed or available.

As this file needs to be read only once, we check whether it was read before. If it was, the command `\captionsenglish` is already defined, so we can stop processing. If this command is undefined we proceed with the various definitions and first show the current version of this file.

```
15.1 ⟨*code⟩
15.2 \ifx\undefined\captionsenglish
15.3 \else
15.4   \selectlanguage{english}
15.5   \expandafter\endinput
15.6 \fi
```

`\atcatcode` This file, `english.ldf`, may have been read while TEX is in the middle of processing a document, so we have to make sure the category code of `@` is 'letter' while this file is being read. We save the category code of the @-sign in `\atcatcode` and make it 'letter'. Later the category code can be restored to whatever it was before.

```
15.7 \chardef\atcatcode=\catcode‘\@
15.8 \catcode‘\@=11\relax
```

Now we determine whether the common macros from the file `babel.def` need to be read. We can be in one of two situations: either another language option has been read earlier on, in which case that other option has already read `babel.def`, or `english` is the first language option to be processed. In that case we need to read `babel.def` right here before we continue.

```
15.9 \ifx\undefined\babel@core@loaded\input babel.def\relax\fi
```

Another check that has to be made, is if another language definition file has been read already. In that case its definitions have been activated. This might interfere with definitions this file tries to make. Therefore we make sure that we cancel any special definitions. This can be done by checking the existence of the macro `\originalTeX`. If it exists we simply execute it, otherwise it is `\let` to `\empty`.

```
15.10 \ifx\undefined\originalTeX \let\originalTeX\empty\fi
15.11 \originalTeX
```

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.

```
15.12 \ifx\undefined\l@english
15.13   \ifx\undefined\l@UKenglish
15.14     \@nopatterns{English}
15.15     \adddialect\l@english0
15.16   \else
15.17     \let\l@english\l@UKenglish
```

[12] The file described in this section has version number ? and was last revised on ?.

```
  \fi
\fi
```

For the American version of these definitions we just add a "dialect". Also, the macros `\captionsamerican` and `\extrasamerican` are `\let` to their English counterparts when these parts are defined.

```
\ifx\l@american\undefined
  \adddialect\l@american\l@english
\fi
```

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.

```
\addto\captionsenglish{%
  \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}%
  }
```

`\captionsamerican` The 'captions' are the same for both versions of the language, so we can `\let` the macro `\captionsamerican` be equal to `\captionsenglish`.

```
\let\captionsamerican\captionsenglish
```

`\dateenglish` The macro `\dateenglish` redefines the command `\today` to produce English dates.

```
\def\dateenglish{%
\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}}
```

`\dateamerican` The macro `\dateamerican` redefines the command `\today` to produce American dates.

```
\def\dateamerican{%
\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}}
```

`\extrasenglish` `\noextrasenglish` The macro `\extrasenglish` will perform all the extra definitions needed for the English language. The macro `\extrasenglish` 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.

```
\addto\extrasenglish{}
\addto\noextrasenglish{}
```

`\extrasamerican` `\noextrasamerican` Also for the "american" variant no extra definitions are needed at the moment.

```
\let\extrasamerican\extrasenglish
\let\noextrasamerican\noextrasenglish
```

It is possible that a site might need to add some extra code to the babel macros. To enable this we load a local configuration file, `english.cfg` if it is found on TEX' search path.

```
\loadlocalcfg{english}
```

Our last action is to make a note that the commands we have just defined, will be executed by calling the macro `\selectlanguage` at the beginning of the document.

```
\main@language{english}
```

Finally, the category code of `@` is reset to its original value. The macrospace used by `\atcatcode` is freed.

```
\catcode`\@=\atcatcode \let\atcatcode\relax
```

⟨/code⟩

16 The German language

The file `germanb.dtx`[13] defines all the language definition macros for the German language as well as for the Austrian dialect of this language[14].

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.

`\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`

[13]The file described in this section has version number ? and was last revised on ?.
[14]This file is a re-implementation of Hubert Partl's `german.sty` version 2.5b, see [4].

As this file, `germanb.ldf`, needs to be read only once, we check whether it was read before. If it was, the command `\captionsgerman` is already defined, so we can stop processing. If this command is undefined we proceed with the various definitions and first show the current version of this file.

```
16.1 ⟨*code⟩
16.2 \ifx\undefined\captionsgerman
16.3 \else
16.4   \selectlanguage{german}
16.5   \expandafter\endinput
16.6 \fi
```

`\atcatcode` This file, `germanb.ldf`, may have been read while TEX is in the middle of processing a document, so we have to make sure the category code of `@` is 'letter' while this file is being read. We save the category code of the @-sign in `\atcatcode` and make it 'letter'. Later the category code can be restored to whatever it was before.

```
16.7 \chardef\atcatcode=\catcode‘\@
16.8 \catcode‘\@=11\relax
```

Now we determine whether the common macros from the file `babel.def` need to be read. We can be in one of two situations: either another language option has been read earlier on, in which case that other option has already read `babel.def`, or `germanb` is the first language option to be processed. In that case we need to read `babel.def` right here before we continue.

```
16.9 \ifx\undefined\babel@core@loaded\input babel.def\relax\fi
```

Another check that has to be made, is if another language definition file has been read already. In that case its definitions have been activated. This might interfere with definitions this file tries to make. Therefore we make sure that we cancel any special definitions. This can be done by checking the existence of the macro `\originalTeX`. If it exists we simply execute it, otherwise it is `\let` to `\empty`.

```
16.10 \ifx\undefined\originalTeX \let\originalTeX\empty\fi
16.11 \originalTeX
```

When this file is read as an option, i.e., by the `\usaepackage` 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.

```
16.12 \ifx\undefined\l@german
16.13   \@nopatterns{German}
16.14   \adddialect\l@german0
16.15 \fi
```

For the Austrian version of these definitions we just add another language. Also, the macros `\captionsaustrian` and `\extrasaustrian` are `\let` to their German counterparts if these parts are defined.

```
16.16 \adddialect\l@austrian\l@german
```

The next step consists of defining commands to switch to (and from) the German language.

\captionsgerman The macro \captionsgerman defines all strings used in the four standard document classes provided with LaTeX.

```
\addto\captionsgerman{%
  \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}%
  }
```

\captionsgerman The 'captions' are the same for both version of the language, so we can \let the macro \captionsaustrian be equal to \captionsgerman.

```
\let\captionsaustrian\captionsgerman
```

\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 The macro \extrasgerman will perform all the extra definitions needed for the
\noextrasgerman 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{"}
\addto\extrasgerman{\languageshorthands{german}}
\addto\extrasgerman{\bbl@activate{"}}
%\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).

```
\addto\extrasgerman{%
  \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\extrasgerman{\babel@save\"\umlautlow}
\addto\noextrasgerman{\umlauthigh}
```

The german hyphenation patterns can be used with \lefthyphenmin and \righthyphenmin set to 2.

```
\def\germanhyphenmins{\tw@\tw@}
```

\extrasaustrian
\noextrasaustrian

For both versions of the language the same special macros are used, so we can \let the austrian macros be equal to their german counterparts.

```
\let\extrasaustrian\extrasgerman
\let\noextrasaustrian\noextrasgerman
```

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
```

\german@dq@disc

For the discretionary macros we use this macro:

```
\def\german@dq@disc#1#2{%
  \penalty\@M\discretionary{#2-}{}{#1}\allowhyphens}
```

Now we can define the doublequote macros: the umlauts,

```
\declare@shorthand{german}{"a}{\textormath{\"{a}}{\ddot a}}
\declare@shorthand{german}{"o}{\textormath{\"{o}}{\ddot o}}
\declare@shorthand{german}{"u}{\textormath{\"{u}}{\ddot u}}
\declare@shorthand{german}{"A}{\textormath{\"{A}}{\ddot A}}
\declare@shorthand{german}{"O}{\textormath{\"{O}}{\ddot O}}
\declare@shorthand{german}{"U}{\textormath{\"{U}}{\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}{"`}{%
  \textormath{\quotedblbase{}}{\mbox{\quotedblbase}}}
\declare@shorthand{german}{"'}{%
  \textormath{\textquotedblleft{}}{\mbox{\textquotedblleft}}}
\declare@shorthand{german}{"<}{%
  \textormath{\guillemotleft{}}{\mbox{\guillemotleft}}}
\declare@shorthand{german}{">}{%
  \textormath{\guillemotright{}}{\mbox{\guillemotright}}}
```

discretionary commands

```
\declare@shorthand{german}{"c}{\textormath{\german@dq@disc ck}{c}}
\declare@shorthand{german}{"C}{\textormath{\german@dq@disc CK}{C}}
\declare@shorthand{german}{"f}{\textormath{\german@dq@disc f{ff}}{f}}
\declare@shorthand{german}{"F}{\textormath{\german@dq@disc F{FF}}{F}}
\declare@shorthand{german}{"l}{\textormath{\german@dq@disc l{ll}}{l}}
\declare@shorthand{german}{"L}{\textormath{\german@dq@disc L{LL}}{L}}
\declare@shorthand{german}{"m}{\textormath{\german@dq@disc m{mm}}{m}}
\declare@shorthand{german}{"M}{\textormath{\german@dq@disc M{MM}}{M}}
\declare@shorthand{german}{"n}{\textormath{\german@dq@disc n{nn}}{n}}
\declare@shorthand{german}{"N}{\textormath{\german@dq@disc N{NN}}{N}}
\declare@shorthand{german}{"p}{\textormath{\german@dq@disc p{pp}}{p}}
\declare@shorthand{german}{"P}{\textormath{\german@dq@disc P{PP}}{P}}
\declare@shorthand{german}{"r}{\textormath{\german@dq@disc r{rr}}{r}}
\declare@shorthand{german}{"R}{\textormath{\german@dq@disc R{RR}}{R}}
\declare@shorthand{german}{"t}{\textormath{\german@dq@disc t{tt}}{t}}
\declare@shorthand{german}{"T}{\textormath{\german@dq@disc T{TT}}{T}}
```

and some additional commands:

```
\declare@shorthand{german}{"-}{\penalty\@M\-\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` All that's left to do now is to define a couple of commands for reasons of compatibility with `german.sty`.

`\ck`

```
\def\mdqon{\bbl@activate{"}}
\def\mdqoff{\bbl@deactivate{"}}
\def\ck{\allowhyphens\discretionary{k-}{k}{ck}\allowhyphens}
```

It is possible that a site might need to add some extra code to the babel macros. To enable this we load a local configuration file, `germanb.cfg` if it is found on TEX' search path.

```
\loadlocalcfg{germanb}
```

Our last action is to make a note that the commands we have just defined, will be executed by calling the macro `\selectlanguage` at the beginning of the document.

16.115 `\main@language{german}`

Finally, the category code of `@` is reset to its original value.

16.116 `\catcode‘\@=\atcatcode`

16.117 ⟨/code⟩

17 The Breton language

The file `breton.dtx`[15] 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.

As this file needs to be read only once, we check whether it was read before. If it was, the `\captionsbreton` is already defined, so we can stop processing. If this command is undefined we proceed with the various definitions and first show the current version of this file.

```
17.1 ⟨*code⟩
17.2 \ifx\undefined\captionsbreton
17.3 \else
17.4   \selectlanguage{breton}
17.5   \expandafter\endinput
17.6 \fi
```

`\atcatcode` This file, `breton.ldf`, may have been read while TEX is in the middle of processing a document, so we have to make sure the category code of `@` is 'letter' while this file is being read. We save the category code of the @-sign in `\atcatcode` and make it 'letter'. Later the category code can be restored to whatever it was before.

```
17.7 \chardef\atcatcode=\catcode‘\@
17.8 \catcode‘\@=11\relax
```

Now we determine whether the the common macros from the file `babel.def` need to be read. We can be in one of two situations: either another language option has been read earlier on, in which case that other option has already read `babel.def`, or `breton` is the first language option to be processed. In that case we need to read `babel.def` right here before we continue.

```
17.9 \ifx\undefined\babel@core@loaded\input babel.def\relax\fi
```

Another check that has to be made, is if another language definition file has been read already. In that case its definitions have been activated. This might interfere with definitions this file tries to make. Therefore we make sure that we cancel any special definitions. This can be done by checking the existence of the macro `\originalTeX`. If it exists we simply execute it, otherwise it is `\let` to `\empty`.

```
17.10 \ifx\undefined\originalTeX \let\originalTeX\empty \fi
17.11 \originalTeX
```

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.

```
17.12 \ifx\undefined\l@breton
17.13     \@nopatterns{Breton}
17.14     \adddialect\l@breton0\fi
```

[15] The file described in this section has version number ? and was last revised on ?.

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}%
  \def\alsoname{Gwelout ivez}%
  \def\proofname{Proof}%  <-- 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 The macro \extrasbreton will perform all the extra definitions needed for the
\noextrasbreton 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{?}}
```

```
%\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}
```

It is possible that a site might need to add some extra code to the babel macros. To enable this we load a local configuration file, `breton.cfg` if it is found on TEX' search path.

```
\loadlocalcfg{breton}
```

Our last action is to make a note that the commands we have just defined, will be executed by calling the macro `\selectlanguage` at the beginning of the document.

```
\main@language{breton}
```

Finally, the category code of `@` is reset to its original value. The macrospace used by `\atcatcode` is freed.

17.123 `\catcode‘\@=\atcatcode \let\atcatcode\relax`

17.124 ⟨/code⟩

18 The Irish language

The file `irish.dtx`[16] defines all the language definition macros for the Irish language.

As this file needs to be read only once, we check whether it was read before. If it was, the command `\captionsirish` is already defined, so we can stop processing. If this command is undefined we proceed with the various definitions and first show the current version of this file.

```
18.1 ⟨*code⟩
18.2 \ifx\undefined\captionsirish
18.3 \else
18.4   \selectlanguage{irish}
18.5   \expandafter\endinput
18.6 \fi
```

`\atcatcode` This file, `irish.ldf`, may have been read while TEX is in the middle of processing a document, so we have to make sure the category code of `@` is 'letter' while this file is being read. We save the category code of the @-sign in `\atcatcode` and make it 'letter'. Later the category code can be restored to whatever it was before.

```
18.7 \chardef\atcatcode=\catcode`\@
18.8 \catcode`\@=11\relax
```

Now we determine whether the the common macros from the file `babel.def` need to be read. We can be in one of two situations: either another language option has been read earlier on, in which case that other option has already read `babel.def`, or `irish` is the first language option to be processed. In that case we need to read `babel.def` right here before we continue.

```
18.9 \ifx\undefined\babel@core@loaded\input babel.def\relax\fi
```

Another check that has to be made, is if another language definition file has been read already. In that case its definitions have been activated. This might interfere with definitions this file tries to make. Therefore we make sure that we cancel any special definitions. This can be done by checking the existence of the macro `\originalTeX`. If it exists we simply execute it.

```
18.10 \ifx\undefined\originalTeX
18.11   \let\originalTeX\empty
18.12 \fi
18.13 \originalTeX
```

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.

```
18.14 \ifx\undefined\l@irish
18.15   \@nopatterns{irish}
18.16   \adddialect\l@irish0\fi
```

The next step consists of defining commands to switch to (and from) the Irish language.

[16] The file described in this section has version number ? and was last revised on ?. A contribution was made by Marion Gunn.

\captionsirish The macro \captionsirish defines all strings used in the four standard documentclasses provided with LaTeX.

```
\addto\captionsirish{%
  \def\prefacename{Preface}%    <-- needs translation
  \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{see}%    <-- needs translation
  \def\alsoname{see also}%    <-- needs translation
  \def\proofname{Proof}%    <-- needs translation
  }
```

\dateirish The macro \dateirish redefines the command \today to produce Irish dates.

```
\def\dateirish{%
  \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 The macro \extrasirish will perform all the extra definitions needed for the \noextrasirish 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{}
```

It is possible that a site might need to add some extra code to the babel macros. To enable this we load a local configuration file, danish.cfg if it is found on TeX' search path.

```
\loadlocalcfg{danish}
```

Our last action is to make a note that the commands we have just defined, will be executed by calling the macro \selectlanguage at the beginning of the document.

```
\main@language{irish}
```

Finally, the category code of @ is reset to its original value. The macrospace used by \atcatcode is freed.

```
\catcode`\@=\atcatcode \let\atcatcode\relax
```
⟨/code⟩

19 The scottish language

The file `scottish.dtx`[17] defines all the language definition macros for the scottish language.

As this file needs to be read only once, we check whether it was read before. If it was, the command `\captionsscottish` is already defined, so we can stop processing. If this command is undefined we proceed with the various definitions and first show the current version of this file.

```
19.1 ⟨*code⟩
19.2 \ifx\undefined\captionsscottish
19.3 \else
19.4   \selectlanguage{scottish}
19.5   \expandafter\endinput
19.6 \fi
```

`\atcatcode` This file, `scottish.ldf`, may have been read while TEX is in the middle of processing a document, so we have to make sure the category code of `@` is 'letter' while this file is being read. We save the category code of the @-sign in `\atcatcode` and make it 'letter'. Later the category code can be restored to whatever it was before.

```
19.7 \chardef\atcatcode=\catcode‘\@
19.8 \catcode‘\@=11\relax
```

Now we determine whether the the common macros from the file `babel.def` need to be read. We can be in one of two situations: either another language option has been read earlier on, in which case that other option has already read `babel.def`, or `scottish` is the first language option to be processed. In that case we need to read `babel.def` right here before we continue.

```
19.9 \ifx\undefined\babel@core@loaded\input babel.def\relax\fi
```

Another check that has to be made, is if another language definition file has been read already. In that case its definitions have been activated. This might interfere with definitions this file tries to make. Therefore we make sure that we cancel any special definitions. This can be done by checking the existence of the macro `\originalTeX`. If it exists we simply execute it.

```
19.10 \ifx\undefined\originalTeX
19.11   \let\originalTeX\empty
19.12 \fi
19.13 \originalTeX
```

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.

```
19.14 \ifx\undefined\l@scottish
19.15   \@nopatterns{scottish}
19.16   \adddialect\l@scottish0\fi
```

The next step consists of defining commands to switch to (and from) the scottish language.

[17] The file described in this section has version number ? and was last revised on ?. A contribution was made by Fraser Grant (`FRASER@CERNVM`).

\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
}
```

\datescottish The macro \datescottish redefines the command \today to produce Scottish dates.

```
\def\datescottish{%
  \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 The macro \extrasscottish will perform all the extra definitions needed for the
\noextrasscottish 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{}
```

It is possible that a site might need to add some extra code to the babel macros. To enable this we load a local configuration file, scottish.cfg if it is found on TeX' search path.

```
\loadlocalcfg{scottish}
```

Our last action is to make a note that the commands we have just defined, will be executed by calling the macro \selectlanguage at the beginning of the document.

```
\main@language{scottish}
```

Finally, the category code of `@` is reset to its original value. The macrospace used by `\atcatcode` is freed.

19.50 `\catcode‘\@=\atcatcode \let\atcatcode\relax`

19.51 ⟨/code⟩

20 The French language

The file `francais.dtx`[18] defines all the language definition macros for the French language.

French typographic rules specify that a little white space should be present before 'double puctuation' characters. These characters are `:`, `;`, `!` and `?`. In order to get this whitespace automatigically the category code of these characters is made `\active`. The user should input these four characters preceeded with a space; the space will then be replaced by a `\thinspace`.

As this file needs to be read only once, we check whether it was read before. If it was, the `\captionsfrancais` is already defined, so we can stop processing. If this command is undefined we proceed with the various definitions and first show the current version of this file.

```
20.1 ⟨*code⟩
20.2 \ifx\undefined\captionsfrancais
20.3 \else
20.4   \selectlanguage{francais}
20.5   \expandafter\endinput
20.6 \fi
```

`\atcatcode` This file, `francais.ldf`, may have been read while TEX is in the middle of processing a document, so we have to make sure the category code of `@` is 'letter' while this file is being read. We save the category code of the @-sign in `\atcatcode` and make it 'letter'. Later the category code can be restored to whatever it was before.

```
20.7 \chardef\atcatcode=\catcode`\@
20.8 \catcode`\@=11\relax
```

Now we determine whether the the common macros from the file `babel.def` need to be read. We can be in one of two situations: either another language option has been read earlier on, in which case that other option has already read `babel.def`, or `francais` is the first language option to be processed. In that case we need to read `babel.def` right here before we continue.

```
20.9 \ifx\undefined\babel@core@loaded\input babel.def\relax\fi
```

Another check that has to be made, is if another language definition file has been read already. In that case its definitions have been activated. This might interfere with definitions this file tries to make. Therefore we make sure that we cancel any special definitions. This can be done by checking the existence of the macro `\originalTeX`. If it exists we simply execute it, otherwise it is `\let` to `\empty`.

```
20.10 \ifx\undefined\originalTeX \let\originalTeX\empty \fi
20.11 \originalTeX
```

When this file is read as an option, i.e. by the `\usepackage` command, `francais` will be an 'unknown' language in which case we have to make it known. So we check for the existence of `\l@francais` to see whether we have to do something here.

[18] 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 French. Later the definitions from `french.sty` version 2 were added.

```
\ifx\l@francais\undefined
  \ifx\l@french\undefined
    \@nopatterns{Francais}
    \adddialect\l@francais0
    \let\l@french\l@francais
  \else
    \let\l@francais\l@french
  \fi
\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.

`\captionsfrancais` The macro `\captionsfrancais` defines all strings used in the four standard document classes provided with LaTeX.

```
\addto\captionsfrancais{%
  \def\prefacename{Pr\'eface}%
  \def\refname{R\'ef\'erences}%
  \def\abstractname{R\'esum\'e}%
  \def\bibname{Bibliographie}%
  \def\chaptername{Chapitre}%
  \def\appendixname{Annexe}%
  \def\contentsname{Table des mati\`eres}%
  \def\listfigurename{Liste des figures}%
  \def\listtablename{Liste des tableaux}%
  \def\indexname{Index}%
  \def\figurename{Figure}%
  \def\tablename{Tableau}%
  \def\partname{Partie}%
  \def\enclname{P.~J.}%
  \def\ccname{Copie \`a}%
  \def\headtoname{A}
  \def\pagename{Page}%
  \def\seename{voir}%
  \def\alsoname{voir aussi}%
  \def\proofname{Proof}%  <-- needs translation!
  }
```

`\datefrancais` The macro `\datefrancais` redefines the command `\today` to produce French dates.

```
\def\datefrancais{%
\def\today{\ifnum\day=1\relax 1\/$^{\rm er}$\else
  \number\day\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}}
```

`\extrasfrancais` `\noextrasfrancais` The macro `\extrasfrancais` will perform all the extra definitions needed for the French language. The macro `\noextrasfrancais` is used to cancel the actions of `\extrasfrancais`.

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 french group of shorthands should be used.

```
\addto\extrasfrancais{\languageshorthands{french}}
```

These characters are 'turned on' once, later their definition may vary.

```
\addto\extrasfrancais{%
  \bbl@activate{:}\bbl@activate{;}%
  \bbl@activate{!}\bbl@activate{?}}
%\addto\noextrasfrancais{%
%  \bbl@deactivate{:}\bbl@deactivate{;}%
%  \bbl@deactivate{!}\bbl@deactivate{?}}
```

The last thing `\extrasfrancais` needs to do is to make sure that `\frenchspacing` is in effect. If this is not the case the execution of `\noextrasfrancais` will switch it off again.

```
\addto\extrasfrancais{\bbl@frenchspacing}
\addto\noextrasfrancais{\bbl@nonfrenchspacing}
```

`\french@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{french}{;}{%
  \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
```

Now we can insert a `;` character.

```
    \string;\space
  \fi}
```

`\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}{:}{%
  \ifmmode\string:\space
  \else\relax
    \ifhmode
      \ifdim\lastskip>\z@\unskip\penalty\@M\thinspace\fi
    \fi
    \string:\space
  \fi}
\declare@shorthand{french}{!}{%
  \ifmmode\string!\space
  \else\relax
```

```
    \ifhmode
      \ifdim\lastskip>\z@\unskip\penalty\@M\thinspace\fi
    \fi
    \string!\space
  \fi}
```

`\french@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{french}{?}{%
  \ifmmode\string?\space
  \else\relax
    \ifhmode
      \ifdim\lastskip>\z@
        \unskip
        \kern\fontdimen2\font
        \kern-1.4\fontdimen3\font
      \fi
    \fi
    \string?\space
  \fi}
```

`\system@sh@:@` When the active characters appear in an environment where their french behaviour `\system@sh@!@` is not wanted they should give an 'expected' result, ie not gobble up the space `\system@sh@?@` that follows them. Therefore we define shorthands at system level as well.

`\system@sh@;@`

```
\declare@shorthand{system}{:}{\string:\space}
\declare@shorthand{system}{!}{\string!\space}
\declare@shorthand{system}{?}{\string?\space}
\declare@shorthand{system}{;}{\string;\space}
```

All that is left to do now is provide the french 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}}
```

A macro for typesetting things like 1^{er} as proposed by Raymon Seroul.

```
\def\up#1{\raise 1ex\hbox{\small#1}}
```

Definitions as provided by Nicolas Brouard for typing `\No3` to get 3° and for typing `4\ieme` to get 4^{e}.

```
\def\No{\kern-.25em\lower.2ex\hbox{\degre}}
\def\ieme{$^{\rm e }$\kern+.17em}
```

And some more macros for numbering. First two support macros.

```
\def\FrenchEnumerate#1{$#1^{\rm o}$\kern+.29em}
\def\FrenchPopularEnumerate#1{#1\No\kern-.25em)\kern+.3em}
```

Typing `\primo` should result in '1^{o} ',

```
\def\primo{\FrenchEnumerate1}
\def\secundo{\FrenchEnumerate2}
\def\tertio{\FrenchEnumerate3}
\def\quatro{\FrenchEnumerate4}
```

while typing `\fprimo)` gives '1°) '.

```
\def\fprimo){\FrenchPopularEnumerate1}
\def\fsecundo){\FrenchPopularEnumerate2}
\def\ftertio){\FrenchPopularEnumerate3}
\def\fquatro){\FrenchPopularEnumerate4}
```

It is possible that a site might need to add some extra code to the babel macros. To enable this we load a local configuration file, `francais.cfg` if it is found on TEX' search path.

```
\loadlocalcfg{francais}
```

Our last action is to make a note that the commands we have just defined, will be executed by calling the macro `\selectlanguage` at the beginning of the document.

```
\main@language{francais}
```

Finally, the category code of `@` is reset to its original value. The macrospace used by `\atcatcode` is freed.

```
\catcode`\@=\atcatcode \let\atcatcode\relax
```

⟨/code⟩

21 The Italian language

The file `italian.dtx`[19] It defines all the language-specific macros for the Italian language.

For this language the `\clubpenalty`, `\widowpenalty` and `\finalhyphendemerits` are set to rather high values.

As this file needs to be read only once, we check whether it was read before. If it was, the command `\captionsitalian` is already defined, so we can stop processing. If this command is undefined we proceed with the various definitions and first show the current version of this file.

```
21.1 ⟨*code⟩
21.2 \ifx\undefined\captionsitalian
21.3 \else
21.4   \selectlanguage{italian}
21.5   \expandafter\endinput
21.6 \fi
```

`\atcatcode` This file, `italian.sty`, may have been read while TEX is in the middle of processing a document, so we have to make sure the category code of `@` is 'letter' while this file is being read. We save the category code of the @-sign in `\atcatcode` and make it 'letter'. Later the category code can be restored to whatever it was before.

```
21.7 \chardef\atcatcode=\catcode‘\@
21.8 \catcode‘\@=11\relax
```

Now we determine whether the the common macros from the file `babel.def` need to be read. We can be in one of two situations: either another language option has been read earlier on, in which case that other option has already read `babel.def`, or `italian` is the first language option to be processed. In that case we need to read `babel.def` right here before we continue.

```
21.9 \ifx\undefined\babel@core@loaded\input babel.def\relax\fi
```

Another check that has to be made, is if another language definition file has been read already. In that case its definitions have been activated. This might interfere with definitions this file tries to make. Therefore we make sure that we cancel any special definitions. This can be done by checking the existence of the macro `\originalTeX`. If it exists we simply execute it, otherwise it is `\let` to `\empty`.

```
21.10 \ifx\undefined\originalTeX \let\originalTeX\empty \fi
21.11 \originalTeX
```

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.

```
21.12 \ifx\undefined\l@italian
21.13     \@nopatterns{Italian}
21.14     \adddialect\l@italian0\fi
```

[19] The file described in this section has version number ? and was last revised on ?. The original author is Maurizio Codogno, (`urcm@ur785.cselt.stet.it`).

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 documentclasses provided with LaTeX.

```
\addto\captionsitalian{%
  \def\prefacename{Prefazione}%
  \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 abbreviation is preferred
  \def\seename{vedi}%
  \def\alsoname{vedi anche}%
  \def\proofname{Proof}%  <-- needs translation
  }
```

\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.

```
\def\italianhyphenmins{\tw@\tw@}
```

\extrasitalian Lower the chance that clubs or widows occur.
\noextrasitalian

```
\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}
```

It is possible that a site might need to add some extra code to the babel macros. To enable this we load a local configuration file, `italian.cfg` if it is found on TEX' search path.

```
\loadlocalcfg{italian}
```

Our last action is to make a note that the commands we have just defined, will be executed by calling the macro `\selectlanguage` at the beginning of the document.

```
\main@language{italian}
```

Finally, the category code of `@` is reset to its original value. The macrospace used by `\atcatcode` is freed.

```
\catcode`\@=\atcatcode \let\atcatcode\relax
⟨/code⟩
```

22 The Portuguese language

The file `portuges.dtx`[20] 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 6 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 6: The extra definitions made by `portuges.ldf`

As this file needs to be read only once, we check whether it was read before. If it was, the command `\captionsportuges` is already defined, so we can stop processing. If this command is undefined we proceed with the various definitions and first show the current version of this file.

```
22.1 ⟨*code⟩
22.2 \ifx\undefined\captionsportuges
22.3 \else
22.4   \selectlanguage{portuges}
22.5   \expandafter\endinput
22.6 \fi
```

`\atcatcode` This file, `portuges.ldf`, may have been read while TEX is in the middle of processing a document, so we have to make sure the category code of `@` is 'letter' while this file is being read. We save the category code of the @-sign in `\atcatcode` and make it 'letter'. Later the category code can be restored to whatever it was before.

```
22.7 \chardef\atcatcode=\catcode‘\@
22.8 \catcode‘\@=11\relax
```

Now we determine whether the the common macros from the file `babel.def` need to be read. We can be in one of two situations: either another language option has been read earlier on, in which case that other option has already read `babel.def`, or `portuges` is the first language option to be processed. In that case we need to read `babel.def` right here before we continue.

```
22.9 \ifx\undefined\babel@core@loaded\input babel.def\relax\fi
```

Another check that has to be made, is if another language definition file has been read already. In that case its definitions have been activated. This might

[20] The file described in this section has version number ? and was last revised on ?. Contributions were made by Jose Pedro Ramalhete (`JRAMALHE@CERNVM` or `Jose-Pedro_Ramalhete@MACMAIL`) and Arnaldo Viegas de Lima `arnaldo@VNET.IBM.COM`.

interfere with definitions this file tries to make. Therefore we make sure that we cancel any special definitions. This can be done by checking the existence of the macro `\originalTeX`. If it exists we simply execute it, otherwise it is `\let` to `\empty`.

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

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.

```
\ifx\undefined\l@portuges
    \@nopatterns{Portuges}
    \adddialect\l@portuges0\fi
```

For the Brasilian version of these definitions we just add a "dialect". Also, the macros `\captionsbrazil` and `\extrasbrazil` are `\let` to their Portuguese counterparts when these parts are defined.

```
\adddialect\l@brazil\l@portuges
```

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.

```
\addto\captionsportuges{%
  \def\prefacename{Pref\'acio}%
  \def\refname{Refer\^encias}%
  \def\abstractname{Resumo}%
  \def\bibname{Bibliografia}%
  \def\chaptername{Cap\'{\i}tulo}%
  \def\appendixname{Ap\^endice}%
  \def\contentsname{\'Indice}%
  \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{Anexos}%
  \def\ccname{C\'opia a}%
  \def\headtoname{Para}%
  \def\pagename{P\'agina}%
  \def\seename{ver}%
  \def\alsoname{ver tamb\'em}%
  \def\proofname{Proof}%  <-- needs translation
  }%
```

`\captionsbrazil` The "captions" are different for both versions of the language, so we define the macro `\captionsbrazil` here.

```
\addto\captionsbrazil{%
  \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}%
\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}%
}
```

`\dateportuges` The macro `\dateportuges` redefines the command `\today` to produce Portuguese dates.

```
\def\dateportuges{%
\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}}
```

`\datebrazil` The macro `\datebrazil` redefines the command `\today` to produce Brasilian dates, for which the names of the months are not capitalized.

```
\def\datebrazil{%
\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}}
```

`\portugueshyphenmins` Set correct values for `\lefthyphenmin` and `\righthyphenmin`.
`\brasilhyphenmins`

```
\def\portugueshyphenmins{\tw@\tw@}
\def\brazilhyphenmins{\tw@\tw@}
```

`\extrasportuges` The macro `\extrasportuges` will perform all the extra definitions needed for the
`\noextrasportuges` 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{"}
\addto\extrasportuges{\languageshorthands{portuges}}
\addto\extrasportuges{\bbl@activate{"}}
%\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 behavew a little different from `\-`.

```
\declare@shorthand{portuges}{"-}{\allowhyphens-\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.

```
\addto\extrasportuges{\babel@save\-}
\addto\extrasportuges{\def\-{\allowhyphens
                             \discretionary{-}{}{}\allowhyphens}}
```

`\ord` `\ro` We also provide an easy way to typeset ordinals, both in the male (`\ord` or `\ro`) and the female (`orda` or `\ra`) form.

`\orda` `\ra`

```
\def\ord{$^{\rm o}$}
\def\orda{$^{\rm a}$}
\let\ro\ord\let\ra\orda
```

`\extrasbrazil` `\noextrasbrazil` Also for the "brazil" variant no extra definitions are needed at the moment.

```
\let\extrasbrazil\extrasportuges
\let\noextrasbrazil\noextrasportuges
```

It is possible that a site might need to add some extra code to the babel macros. To enable this we load a local configuration file, `portuges.cfg` if it is found on TEX' search path.

```
\loadlocalcfg{portuges}
```

Our last action is to make a note that the commands we have just defined, will be executed by calling the macro `\selectlanguage` at the beginning of the document.

```
\main@language{portuges}
```

Finally, the category code of `@` is reset to its original value. The macrospace used by `\atcatcode` is freed.

```
\catcode`\@=\atcatcode \let\atcatcode\relax
⟨/code⟩
```

23 The Spanish language

The file `spanish.dtx`[21] defines all the language definition macro's for the Spanish[22] language.

This file[23] incorporates the result of discussions held in the Spanish-TEX[24] electronic mail list.

For this language the characters `'` `~` and `"` are made active. In table 7 an overview is given of their purpose. These active accent characters behave according

`'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.
`"|`	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."
`\-`	like the old `\-`, but allowing hyphenation in the rest of the word.
`"u`	a u with dieresis allowing hyphenation.
`"a`	feminine ordinal as in 1ª.
`"o`	masculine ordinal as in 1º.
`"<`	for French left double quotes (similar to <<).
`">`	for French right double quotes (similar to >>).
`~n`	a n with tilde. Works for uppercase too.

Table 7: The extra definitions made by `spanish.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. Old versions of this file based this support on the existance of special macros with names as in Ferguson's ML-TEX. This is no longer the case. Support is now based on providing an appropriate definition for the accent macros on entry to the Spanish language. This is automatically done by LATEX 2_ε or NFSS2. If T1 encoding is chosen, and provided that adequate hyphenation patterns[25] exist, it

[21] The file described in this section has version number ? and was last revised on ?. The original author is Julio Sánchez, (`jsanchez@gmv.es`).

[22] Catalan used to be part of this file but is now on its own file.

[23] In writing this file, many ideas and actual coding solutions have been taken from a number of sources. The language definition files `dutch.sty` and `germanb.sty` are the main contributors and are not explicitly mentioned in the sequel. J. L. Braams and Bernd Raichle have given helpful advice. Another source of inspiration is the experience gained in the use of FTC, a software package written by José A. Mañas. The members of the Spanish-TEX list have helped clarify a number of issues. Other sources are explicitly acknowledged when used. If you think that you contributed something and you are not mentioned, please let me (`jsanchez@gmv.es`) know. I humbly apologize for any omission.

[24] `spanish-tex@goya.eunet.es`, subscription requests can be sent to the address `listserv@goya.eunet.es`. This list is devoted to discussions on support in TEX for Spanish. Comments on this language option are welcome there or directly to `jsanchez@gmv.es`.

[25] One source for such patterns is the archive at `ftp.eunet.es` that can be accessed by anonymous FTP or electronic mail to `ftpmail@goya.eunet.es`. They are in the `info` direc-

is possible to get better hyphenation for Spanish than before. The easiest way to use the new encoding with LaTeX 2$_\varepsilon$ to load the package `t1enc` with `\usepackage`. This must be done before loading babel.

If the combination of keyboard and TeX version that the user has is able to produce the accented characters in the T1 enconding, the user could see the accented characters in the editor, greatly improving the readability of the document source. As of today, this is not a recommended method for producing documents for distribution, although it is possible to mechanically translate the document so that the receiver can make use of it. If care is taken to define the encoding needed by the document, the results are pretty portable.

This option file will automatically detect if the T1 encoding is being used and behave appropriately. If any other encoding is being used, the accent macros will be redefined to allow hyphenation on the accented words.

As this file needs to be read only once, we check whether it was read before. If it was, the `\captionsspanish` is already defined, so we can stop processing. If this command is undefined we proceed with the various definitions and first show the current version of this file.

```
23.1 ⟨*code⟩
23.2 \ifx\undefined\captionsspanish
23.3 \else
23.4   \selectlanguage{spanish}
23.5   \expandafter\endinput
23.6 \fi
```

`\atcatcode` This file, `spanish.ldf`, may have been read while TeX is in the middle of processing a document, so we have to make sure the category code of `@` is 'letter' while this file is being read. We save the category code of the @-sign in `\atcatcode` and make it 'letter'. Later the category code can be restored to whatever it was before.

```
23.7 \chardef\atcatcode=\catcode‘\@
23.8 \catcode‘\@=11\relax
```

Now we determine whether the common macros from the file `babel.def` need to be read. We can be in one of two situations: either another language option has been read earlier on, in which case that other option has already read `babel.def`, or `spanish` is the first language option to be processed. In that case we need to read `babel.def` right here before we continue.

```
23.9 \ifx\undefined\babel@core@loaded\input babel.def\relax\fi
```

Another check that has to be made, is if another language definition file has been read already. In that case its definitions have been activated. This might interfere with definitions this file tries to make. Therefore we make sure that we cancel any special definitions. This can be done by checking the existence of the macro `\originalTeX`. If it exists we simply execute it, otherwise it is `\let` to `\empty`.

```
23.10 \ifx\undefined\originalTeX \let\originalTeX\empty \else\originalTeX\fi
```

tory `src/TeX/spanish`. The list of Frequently Asked Questions with Answers about TeX for Spanish is kept there as well. That list is meant to be a summary of the discussions held in the Spanish-TeX mail list. Warning: It is in Spanish.

When this file is read as an option, i.e. by the `\usepackage` command, `spanish` could 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.

```
\ifx\undefined\l@spanish
  \@nopatterns{Spanish}
  \adddialect\l@spanish0
\fi
```

The next step consists of defining commands to switch to (and from) the Spanish language.

`\captionsspanish` The macro `\captionsspanish` defines all strings[26] used in the four standard documentclasses provided with LaTeX.

```
\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\contentsname{\'Indice General}%
  \def\listfigurename{\'Indice de Figuras}%
  \def\listtablename{\'Indice de Tablas}%
  \def\indexname{\'Indice de Materias}%
  \def\figurename{Figura}%
  \def\tablename{Tabla}%
  \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{Proof}%  <-- needs translation!
  }%
```

`\datespanish` The macro `\datespanish` redefines the command `\today` to produce Spanish[27] dates.

```
\def\datespanish{%
\def\today{\number\day~de\space\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
  \space de~\number\year}}
```

`\extrasspanish` `\noextrasspanish` The macro `\extrasspanish` will perform all the extra definitions needed for the Spanish language. The macro `\noextrasspanish` is used to cancel the actions of

[26] The accent on the uppercase 'I' is intentional, following the recommendation of the *Real Academia de la Lengua* in *Esbozo de una Nueva Gramática de la Lengua Española, Comisión de Gramática, Espasa-Calpe, 1973.*

[27] Months are written lowercased. This has been cause of some controversy. This file follows *Diccionario de Uso de la Lengua Española, María Moliner, 1990,* that is in agreement with the most common practice.

\extrasspanish. For Spanish, some characters are made active or are redefined. In particular, the " character, the ’ character and the ~ character receive new meanings. Therefore these characters have to be treated as ‘special’ characters.

```
\addto\extrasspanish{\languageshorthands{spanish}}
\initiate@active@char{"}
\initiate@active@char{~}
\addto\extrasspanish{%
  \bbl@activate{"}%
  \bbl@activate{~}}
\@ifpackagewith{babel}{activeacute}{%
  \initiate@active@char{’}
  \addto\extrasspanish{\bbl@activate{’}}}{}
%\addto\noextrasspanish{
%  \bbl@deactivate{"}\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\extrasspanish{%
  \babel@save\"
  \babel@save\~
  \def\"{\protect\@umlaut}%
  \def\~{\protect\@tilde}}
\@ifpackagewith{babel}{activeacute}{%
  \babel@save\’
  \addto\extrasspanish{\def\’{\protect\@acute}}
  }{}
```

\spanishhyphenmins Spanish hyphenation uses \lefthyphenmin and \righthyphenmin both set to 2.

```
\def\spanishhyphenmins{\tw@\tw@}
```

\dieresis \textacute \texttilde The original definition of \" is stored as \dieresis, 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 \acute or \tilde as the macro names because they are already defined as \mathaccent.

```
\let\dieresis\"
\let\texttilde\~
\@ifpackagewith{babel}{activeacute}{\let\textacute\’}{}
```

\@umlaut \@acute \@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\undefined\DeclareFontShape
  \wlog{Warning: You are using an old LaTeX}
  \wlog{Some word breaks will not be found.}
  \def\@umlaut#1{\allowhyphens\dieresis{#1}\allowhyphens}
  \def\@tilde#1{\allowhyphens\texttilde{#1}\allowhyphens}
  \@ifpackagewith{babel}{activeacute}{%
    \def\@acute#1{\allowhyphens\textacute{#1}\allowhyphens}}{}
\else
  \edef\next{T1}
  \ifx\f@encoding\next
    \let\@umlaut\dieresis
    \let\@tilde\texttilde
    \@ifpackagewith{babel}{activeacute}{%
      \let\@acute\textacute}{}
  \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}
    \def\@tilde#1{\allowhyphens\texttilde{#1}\allowhyphens}
    \@ifpackagewith{babel}{activeacute}{%
      \def\@acute#1{\allowhyphens\textacute{#1}\allowhyphens}}{}
  \fi
\fi
```

Now we can define our shorthands: the umlauts,

```
\declare@shorthand{spanish}{"u}{\@umlaut u}
\declare@shorthand{spanish}{"U}{\@umlaut U}
```

french quotes,

```
\declare@shorthand{spanish}{"<}{%
  \textormath{\guillemotleft{}}{\mbox{\guillemotleft}}}
\declare@shorthand{spanish}{">}{%
  \textormath{\guillemotright{}}{\mbox{\guillemotright}}}
```

ordinals[28],

```
\declare@shorthand{spanish}{"o}{%
  \raise1ex\hbox{\underbar{\scriptsize o}}}
\declare@shorthand{spanish}{"a}{%
  \raise1ex\hbox{\underbar{\scriptsize a}}}
```

acute accents,

```
\@ifpackagewith{babel}{activeacute}{%
  \declare@shorthand{spanish}{'a}{\textormath{\@acute a}{^{\prime} a}}
  \declare@shorthand{spanish}{'e}{\textormath{\@acute e}{^{\prime} e}}
  \declare@shorthand{spanish}{'i}{\textormath{\@acute \i{}}{^{\prime} i}}
  \declare@shorthand{spanish}{'o}{\textormath{\@acute o}{^{\prime} o}}
  \declare@shorthand{spanish}{'u}{\textormath{\@acute u}{^{\prime} u}}
  \declare@shorthand{spanish}{'A}{\textormath{\@acute A}{^{\prime} A}}
  \declare@shorthand{spanish}{'E}{\textormath{\@acute E}{^{\prime} E}}
```

[28] 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.

```
  \declare@shorthand{spanish}{'I}{\textormath{\@acute I}{^{\prime} I}}
  \declare@shorthand{spanish}{'O}{\textormath{\@acute O}{^{\prime} O}}
  \declare@shorthand{spanish}{'U}{\textormath{\@acute U}{^{\prime} U}}
```

the acute accent,

```
  \declare@shorthand{spanish}{''}{%
    \textormath{\textquotedblright}{\sp\bgroup\prim@s'}}
```

tildes,

```
  \declare@shorthand{spanish}{'n}{\textormath{\~n}{^{\prime} n}}
  \declare@shorthand{spanish}{'N}{\textormath{\~N}{^{\prime} N}}
  }{}
\declare@shorthand{spanish}{~n}{\textormath{\~n}{\@tilde n}}
\declare@shorthand{spanish}{~N}{\textormath{\~N}{\@tilde N}}
```

and some additional commands:

```
\declare@shorthand{spanish}{"-}{\allowhyphens\-\allowhyphens}
\declare@shorthand{spanish}{"|}{%
  \textormath{\penalty\@M\discretionary{-}{}{\kern.03em}%
              \allowhyphens}{}}
\declare@shorthand{spanish}{""}{\hskip\z@skip}
```

It is possible that a site might need to add some extra code to the babel macros. To enable this we load a local configuration file, `spanish.cfg` if it is found on TEX' search path.

```
\loadlocalcfg{spanish}
```

Next the babel macro `\main@language` is used to activate the definitions for Spanish at the beginning of the document.

```
\main@language{spanish}
```

Finally, the category code of `@` is reset to its original value. The macrospace used by `\atcatcode` is freed.

```
\catcode`\@=\atcatcode \let\atcatcode\relax
```

⟨/code⟩

24 The Catalan language

The file `catalan.dtx`[29] 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 8 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 9 become also available[30].

`\lgem`	geminated-l digraph (similar to l·l). `\Lgem` produces the uppercase version.
`\up`	Macro to help typing raised ordinals, like 1^{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 8: Extra definitions made by file `catalan.ldf` (activated by default)

`'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 9: Extra definitions made by file `catalan.ldf` (activated only when using the options activeacute and activegrave)

These active accents characters behave according to their original definitions if not followed by one of the characters indicated in that table.

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

[29] The file described in this section has version number ? and was last revised on ?.

[30] 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, for example, to type `l'{}{}astre` instead of `l'astre`.

processing. If this command is undefined we proceed with the various definitions and first show the current version of this file.

```
24.1 ⟨*code⟩
24.2 \ifx\undefined\captionscatalan
24.3 \else
24.4   \selectlanguage{catalan}
24.5   \expandafter\endinput
24.6 \fi
```

\atcatcode This file, `catalan.ldf`, may have been read while TEX is in the middle of processing a document, so we have to make sure the category code of `@` is 'letter' while this file is being read. We save the category code of the @-sign in `\atcatcode` and make it 'letter'. Later the category code can be restored to whatever it was before.

```
24.7 \chardef\atcatcode=\catcode‘\@
24.8 \catcode‘\@=11\relax
```

Now we determine whether the common macros from the file `babel.def` need to be read. We can be in one of two situations: either another language option has been read earlier on, in which case that other option has already read `babel.def`, or `catalan` is the first language option to be processed. In that case we need to read `babel.def` right here before we continue.

```
24.9 \ifx\undefined\babel@core@loaded\input babel.def\fi
```

Another check that has to be made, is if another language definition file has been read already. In that case its definitions have been activated. This might interfere with definitions this file tries to make. Therefore we make sure that we cancel any special definitions. This can be done by checking the existence of the macro `\originalTeX`. If it exists we simply execute it, otherwise it is `\let` to `\empty`.

```
24.10 \ifx\undefined\originalTeX
24.11   \let\originalTeX\empty
24.12 \fi
24.13 \originalTeX
```

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 check for the existence of `\l@catalan` to see whether we have to do something here.

```
24.14 \ifx\undefined\l@catalan
24.15   \@nopatterns{Catalan}
24.16   \adddialect\l@catalan0
24.17 \fi
```

The next step consists of defining commands to switch to (and from) the Catalan language.

\captionscatalan The macro `\captionscatalan` defines all strings used in the four standard documentclasses provided with LATEX.

```
24.18 \addto\captionscatalan{%
24.19   \def\prefacename{Prefaci}%
24.20   \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 de mat\`eries}%
  \def\figurename{Figura}%
  \def\tablename{Taula}%
  \def\partname{Part}%
  \def\enclname{Adjunt}%
  \def\ccname{C\`opia a}%
  \def\headtoname{A}%
  \def\pagename{P\`agina}%
  \def\seename{veure}%
  \def\alsoname{veure tamb\'e}%
  \def\proofname{Demostraci\'o}%
}
```

`\datecatalan` The macro `\datecatalan` redefines the command `\today` to produce Catalan dates. Months are written in lowercase[31].

```
\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`.

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{"}}
\@ifpackagewith{babel}{activegrave}{%
  \initiate@active@char{`}
  \addto\extrascatalan{\bbl@activate{`}}}{}
\@ifpackagewith{babel}{activeacute}{%
  \initiate@active@char{'}
  \addto\extrascatalan{\bbl@activate{'}}}{}
%\addto\noextrascatalan{%
%  \bbl@deactivate{"}
%  \bbl@deactivate{`}\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.

[31] This seems to be the common practice. See for example: E. Coromina, *El 9 Nou: Manual de redacció i estil*, Ed. Eumo, Vic, 1993

We provide new definitions for the accent macros when one or boith 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 8 and 9.

\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
  \edef\next{T1}
  \ifx\f@encoding\next
    \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.[32]

Now we can define our shorthands: the diaeresis and "ela geminada" support,

```
\declare@shorthand{catalan}{"i}{\textormath{\@umlaut\i}{\ddot\imath}}
\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}}
```

[32] 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}{`O}{\textormath{\@grave O}{\grave O}}
  }{}
```

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}}
```

the acute accent,

```
  \declare@shorthand{catalan}{''}{%
    \textormath{\textquotedblright}{\sp\bgroup\prim@s'}}
  }{}
```

and finally, some support definitions

```
\declare@shorthand{catalan}{"-}{\allowhyphens-\allowhyphens}
\declare@shorthand{catalan}{"|}{%
  \textormath{\penalty\@M\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\-{\allowhyphens\discretionary{-}{}{}\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[33].

The code used in the actual macro used is a combination of the one proposed by Feruglio and Fuster[34] and the proposal from Valiente to appear in the next TEX Users Group Annual Meeting (1995). This last proposal has not been fully implemented due to its limitation to CM fonts.

```
\newskip\zzz
\newdimen\leftllkern \newdimen\rightllkern \newdimen\raiselldim
\def\lgem{\relax\ifmmode\orig@ll
  \else
```

[33] The macro names `\ll` and `\LL` were not taken because of the fact that `\ll` is already used in mathematical mode.

[34] G.V. Valiente and R. Fuster, Typesetting Catalan Texts with TEX, *TUGboat* **14**(3), 1993.

```
    \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
    }
\def\Lgem{\leftllkern=0pt\rightllkern=0pt\raiselldim=0pt%
  \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}
```

`\up` A macro for typesetting things like 1^{er} as proposed by Raymon Seroul[35].

```
\def\up#1{\raise 1ex\hbox{\small#1}}
```

It is possible that a site might need to add some extra code to the babel macros. To enable this we load a local configuration file, `catalan.cfg` if it is found on TEX' search path.

```
\loadlocalcfg{catalan}
```

Our last action is to make a note that the commands we have just defined, will be executed by calling the macro `\selectlanguage` at the beginning of the document.

```
\main@language{catalan}
```

Finally, the category code of `@` is reset to its original value. The macrospace used by `\atcatcode` is freed.

```
\catcode`\@=\atcatcode \let\atcatcode\relax
⟨/code⟩
```

[35] This macro has been borrowed from francais.dtx

25 The Galician language

The file `galician.dtx`[36] defines all the language definition macros for the Galician language.

For this language the characters `' ~` and `"` are made active. In table 10 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 1ª.
`"o`	masculine ordinal as in 1º.
`~n`	a n with tilde. Works for uppercase too.

Table 10: The extra definitions made by `galician.ldf`

to their original definitions if not followed by one of the characters indicated in that table.

As this file needs to be read only once, we check whether it was read before. If it was, the command `\captionsgalician` is already defined, so we can stop processing. If this command is undefined we proceed with the various definitions and first show the current version of this file.

```
⟨*code⟩
\ifx\undefined\captionsgalician
\else
  \selectlanguage{galician}
  \expandafter\endinput
\fi
```

`\atcatcode` This file, `galician.ldf`, may have been read while TEX is in the middle of processing a document, so we have to make sure the category code of `@` is 'letter' while this file is being read. We save the category code of the @-sign in `\atcatcode` and make it 'letter'. Later the category code can be restored to whatever it was before.

```
\chardef\atcatcode=\catcode`\@
\catcode`\@=11\relax
```

Now we determine whether the the common macros from the file `babel.def` need to be read. We can be in one of two situations: either another language option has been read earlier on, in which case that other option has already read `babel.def`, or `galician` is the first language option to be processed. In that case we need to read `babel.def` right here before we continue.

```
\ifx\undefined\babel@core@loaded\input babel.def\relax\fi
```

[36] The file described in this section has version number ? and was last revised on ?.

Another check that has to be made, is if another language definition file has been read already. In that case its definitions have been activated. This might interfere with definitions this file tries to make. Therefore we make sure that we cancel any special definitions. This can be done by checking the existence of the macro `\originalTeX`. If it exists we simply execute it.

```
\ifx\undefined\originalTeX
  \let\originalTeX\empty
\fi
\originalTeX
```

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.

```
\ifx\undefined\l@galician
  \@nopatterns{Galician}
  \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.

```
\addto\captionsgalician{%
  \def\prefacename{Prefacio}%
  \def\refname{Referencias}%
  \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{Proof}%  <-- Needs Translation!
}
```

`\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{’}
  \addto\extrasgalician{\bbl@activate{’}}}{}
%\addto\noextrasgalician{%
%  \bbl@deactivate{"}\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\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 10.

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.[37]

```
\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}{"-}{\allowhyphens-\allowhyphens}
\declare@shorthand{galician}{"|}{\discretionary{-}{}{\kern.03em}}
\declare@shorthand{galician}{"u}{\@umlaut{u}}
\declare@shorthand{galician}{"U}{\@umlaut{U}}
```

ordinals[38],

```
\declare@shorthand{galician}{"o}{%
  \raise1ex\hbox{\underbar{\scriptsize o}}}
\declare@shorthand{galician}{"a}{%
  \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}}
```

[37] 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.

[38] 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`.

```
  }{}
\declare@shorthand{galician}{~n}{\textormath{\~n}{\@tilde n}}
\declare@shorthand{galician}{~N}{\textormath{\~N}{\@tilde N}}
```

`\-` 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\-{\allowhyphens\discretionary{-}{}{}\allowhyphens}}
```

It is possible that a site might need to add some extra code to the babel macros. To enable this we load a local configuration file, `galician.cfg` if it is found on TEX' search path.

```
\loadlocalcfg{galician}
```

Our last action is to make a note that the commands we have just defined, will be executed by calling the macro `\selectlanguage` at the beginning of the document.

```
\main@language{galician}
```

Finally, the category code of `@` is reset to its original value. The macrospace used by `\atcatcode` is freed.

```
\catcode`\@\atcatcode \let\atcatcode\relax
⟨/code⟩
```

26 The Romanian language

The file `romanian.dtx`[39] defines all the language-specific macros for the Romanian language.

For this language currently no special definitions are needed or available.

As this file needs to be read only once, we check whether it was read before. If it was, the command `\captionsromanian` is already defined, so we can stop processing. If this command is undefined we proceed with the various definitions and first show the current version of this file.

```
26.1 ⟨*code⟩
26.2 \ifx\undefined\captionsromanian
26.3 \else
26.4   \selectlanguage{romanian}
26.5   \expandafter\endinput
26.6 \fi
```

`\atcatcode` This file, `romanian.sty`, may have been read while TEX is in the middle of processing a document, so we have to make sure the category code of `@` is 'letter' while this file is being read. We save the category code of the @-sign in `\atcatcode` and make it 'letter'. Later the category code can be restored to whatever it was before.

```
26.7 \chardef\atcatcode=\catcode‘\@
26.8 \catcode‘\@=11\relax
```

Now we determine whether the the common macros from the file `babel.def` need to be read. We can be in one of two situations: either another language option has been read earlier on, in which case that other option has already read `babel.def`, or `romanian` is the first language option to be processed. In that case we need to read `babel.def` right here before we continue.

```
26.9 \ifx\undefined\babel@core@loaded\input babel.def\relax\fi
```

Another check that has to be made, is if another language definition file has been read already. In that case its definitions have been activated. This might interfere with definitions this file tries to make. Therefore we make sure that we cancel any special definitions. This can be done by checking the existence of the macro `\originalTeX`. If it exists we simply execute it, otherwise it is `\let` to `\empty`.

```
26.10 \ifx\undefined\originalTeX \let\originalTeX\empty \else\originalTeX\fi
```

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.

```
26.11 \ifx\undefined\l@romanian
26.12     \@nopatterns{Romanian}
26.13     \adddialect\l@romanian0\fi
```

The next step consists of defining commands to switch to (and from) the Romanian language.

[39] The file described in this section has version number ? and was last revised on ?. A contribution was made by Umstatter Horst (`hhu@cernvm.cern.ch`).

\captionsromanian The macro \captionsromanian defines all strings used in the four standard documentlasses provided with LaTeX.

```
\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{Proof}%   <-- needs translation
  }%
```

\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 The macro \extrasromanian will perform all the extra definitions needed for the
\noextrasromanian 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{}
```

It is possible that a site might need to add some extra code to the babel macros. To enable this we load a local configuration file, romanian.cfg if it is found on TeX' search path.

```
\loadlocalcfg{romanian}
```

Our last action is to make a note that the commands we have just defined, will be executed by calling the macro \selectlanguage at the beginning of the document.

```
\main@language{romanian}
```

Finally, the category code of @ is reset to its original value. The macrospace used by \atcatcode is freed.

```
\catcode`\@=\atcatcode \let\atcatcode\relax
```
⟨/code⟩

27 The Danish language

The file `danish.dtx`[40] defines all the language definition macros for the Danish language.

For this language the character " is made active. In table 11 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 11: The extra definitions made by `danish.ldf`

As this file needs to be read only once, we check whether it was read before. If it was, the command `\captionsdanish` is already defined, so we can stop processing. If this command is undefined we proceed with the various definitions and first show the current version of this file.

```
27.1 ⟨*code⟩
27.2 \ifx\undefined\captionsdanish
27.3 \else
27.4   \selectlanguage{danish}
27.5   \expandafter\endinput
27.6 \fi
```

`\atcatcode` This file, `danish.sty`, may have been read while TEX is in the middle of processing a document, so we have to make sure the category code of `@` is 'letter' while this file is being read. We save the category code of the @-sign in `\atcatcode` and make it 'letter'. Later the category code can be restored to whatever it was before.

```
27.7 \chardef\atcatcode=\catcode‘\@
27.8 \catcode‘\@=11\relax
```

Now we determine whether the the common macros from the file `babel.def` need to be read. We can be in one of two situations: either another language option has been read earlier on, in which case that other option has already read `babel.def`, or `danish` is the first language option to be processed. In that case we need to read `babel.def` right here before we continue.

```
27.9 \ifx\undefined\babel@core@loaded\input babel.def\relax\fi
```

Another check that has to be made, is if another language definition file has been read already. In that case its definitions have been activated. This might interfere with definitions this file tries to make. Therefore we make sure that we cancel any special definitions. This can be done by checking the existence of the

[40] The file described in this section has version number ? and was last revised on ?. A contribution was made by Henning Larsen (`larsen@cernvm.cern.ch`)

macro \originalTeX. If it exists we simply execute it, otherwise it is \let to \empty.

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

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.

```
\ifx\undefined\l@danish
    \@nopatterns{Danish}
    \adddialect\l@danish0\fi
```

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.

```
\addto\captionsdanish{%
  \def\prefacename{Forord}%
  \def\refname{Litteratur}%
  \def\abstractname{Resum\'e}%
  \def\bibname{Litteratur}%
  \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{Proof}%  <-- needs translation!
  }%
```

\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 The macro \extrasdanish will perform all the extra definitions needed for the
\noextrasdanish Danish language. The macro \noextrasdanish is used to cancel the actions of \extrasdanish.

Danish typesetting requires \frencspacing 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{"}}
%\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{\textquotedblright{}}{\mbox{\textquotedblright}}}
\declare@shorthand{danish}{"<}{%
  \textormath{\guillemotleft{}}{\mbox{\guillemotleft}}}
\declare@shorthand{danish}{">}{%
  \textormath{\guillemotright{}}{\mbox{\guillemotright}}}
```

then we define two shorthands to be able to specify hyphenation breakpoints that behavew a little different from `\-`.

```
\declare@shorthand{danish}{"-}{\allowhyphens-\allowhyphens}
\declare@shorthand{danish}{""}{\hskip\z@skip}
```

And we want to have a shorthand for disabling a ligature.

```
\declare@shorthand{danish}{"|}{%
  \textormath{\discretionary{-}{}{\kern.03em}}{}}
```

It is possible that a site might need to add some extra code to the babel macros. To enable this we load a local configuration file, `danish.cfg` if it is found on TEX' search path.

```
\loadlocalcfg{danish}
```

Our last action is to make a note that the commands we have just defined, will be executed by calling the macro `\selectlanguage` at the beginning of the document.

```
\main@language{danish}
```

Finally, the category code of `@` is reset to its original value. The macrospace used by `\atcatcode` is freed.

```
\catcode`\@=\atcatcode \let\atcatcode\relax
```

⟨/code⟩

28 The Norwegian language

The file `norsk.dtx`[41] defines all the language definition macros for the Norwegian language as well as for a new spelling variant 'nynorsk' for this language.

For this language currently no special definitions are needed or available.

As this file needs to be read only once, we check whether it was read before. If it was, the command `\captionsnorsk` is already defined, so we can stop processing. If this command is undefined we proceed with the various definitions and first show the current version of this file.

```
28.1 ⟨*code⟩
28.2 \ifx\undefined\captionsnorsk
28.3 \else
28.4   \selectlanguage{norsk}
28.5   \expandafter\endinput
28.6 \fi
```

`\atcatcode` This file, `norsk.sty`, may have been read while TEX is in the middle of processing a document, so we have to make sure the category code of `@` is 'letter' while this file is being read. We save the category code of the @-sign in `\atcatcode` and make it 'letter'. Later the category code can be restored to whatever it was before.

```
28.7 \chardef\atcatcode=\catcode`\@
28.8 \catcode`\@=11\relax
```

Now we determine whether the the common macros from the file `babel.def` need to be read. We can be in one of two situations: either another language option has been read earlier on, in which case that other option has already read `babel.def`, or `norsk` is the first language option to be processed. In that case we need to read `babel.def` right here before we continue.

```
28.9 \ifx\undefined\babel@core@loaded\input babel.def\relax\fi
```

Another check that has to be made, is if another language definition file has been read already. In that case its definitions have been activated. This might interfere with definitions this file tries to make. Therefore we make sure that we cancel any special definitions. This can be done by checking the existence of the macro `\originalTeX`. If it exists we simply execute it, otherwise it is `\let` to `\empty`.

```
28.10 \ifx\undefined\originalTeX \let\originalTeX\empty \else\originalTeX\fi
```

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.

```
28.11 \ifx\undefined\l@norsk
28.12     \@nopatterns{Norsk}
28.13     \adddialect\l@norsk0\fi
```

For the 'nynorsk' version of these definitions we just add a "dialect". Also, the macros `\datenynorsk` and `\extrasnynorsk` are `\let` to their 'norsk' counterparts when these parts are defined.

```
28.14 \adddialect\l@nynorsk\l@norsk
```

[41] The file described in this section has version number ? and was last revised on ?. 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`).

\norskhyphenmins
\nynorskhyphenmins

The Norwegian hyphenation patterns can be used with \lefthyphenmin set to 1 and \righthyphenmin set to 2. This is true for both 'versions' of the language.

```
\def\norskhyphenmins{\@ne\tw@}
\let\nynorskhyphenmins\norskhyphenmins
```

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.

```
\addto\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{Proof}%
  }
```

\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.

```
\addto\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 sendt}%
\def\headtoname{Til}% in letter
\def\pagename{Side}%
\def\seename{Sj\aa{}}%
\def\alsoname{Sj\aa{} ogs\aa{}}%
\def\proofname{Proof}%
}
```

`\datenorsk` The macro `\datenorsk` redefines the command `\today` to produce Norwegian dates.

```
\def\datenorsk{%
\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}}
```

`\datenynorsk` The spelling of the names of the months is the same for both versions of the "Norsk" language, so we simply `\let` the macro `\datenynorsk` be equal to `\datenorsk`

```
\let\datenynorsk\datenorsk
```

`\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.

```
\addto\extrasnorsk{\bbl@frenchspacing}
\addto\noextrasnorsk{\bbl@nonfrenchspacing}
```

`\extrasnynorsk` `\noextrasnynorsk` Also for the "nynorsk" variant no extra definitions are needed at the moment.

```
\let\extrasnynorsk\extrasnorsk
\let\noextrasnynorsk\noextrasnorsk
```

It is possible that a site might need to add some extra code to the babel macros. To enable this we load a local configuration file, `norsk.cfg` if it is found on TEX' search path.

```
\loadlocalcfg{norsk}
```

Our last action is to make a note that the commands we have just defined, will be executed by calling the macro `\selectlanguage` at the beginning of the document.

```
\main@language{norsk}
```

Finally, the category code of `@` is reset to its original value. The macrospace used by `\atcatcode` is freed.

```
\catcode`\@=\atcatcode \let\atcatcode\relax
⟨/code⟩
```

29 The Swedish language

The file `swedish.dtx`[42] defines all the language-specific macros for the Swedish language.

For this language currently no special definitions are needed or available.

As this file needs to be read only once, we check whether it was read before. If it was, the command `\captionsswedish` is already defined, so we can stop processing. If this command is undefined we proceed with the various definitions and first show the current version of this file.

```
29.1 ⟨*code⟩
29.2 \ifx\undefined\captionsswedish
29.3 \else
29.4   \selectlanguage{swedish}
29.5   \expandafter\endinput
29.6 \fi
```

`\atcatcode` This file, `swedish.sty`, may have been read while TEX is in the middle of processing a document, so we have to make sure the category code of `@` is 'letter' while this file is being read. We save the category code of the @-sign in `\atcatcode` and make it 'letter'. Later the category code can be restored to whatever it was before.

```
29.7 \chardef\atcatcode=\catcode`\@
29.8 \catcode`\@=11\relax
```

Now we determine whether the the common macros from the file `babel.def` need to be read. We can be in one of two situations: either another language option has been read earlier on, in which case that other option has already read `babel.def`, or `swedish` is the first language option to be processed. In that case we need to read `babel.def` right here before we continue.

```
29.9 \ifx\undefined\babel@core@loaded\input babel.def\relax\fi
```

Another check that has to be made, is if another language definition file has been read already. In that case its definitions have been activated. This might interfere with definitions this file tries to make. Therefore we make sure that we cancel any special definitions. This can be done by checking the existence of the macro `\originalTeX`. If it exists we simply execute it, otherwise it is `\let` to `\empty`.

```
29.10 \ifx\undefined\originalTeX \let\originalTeX\empty \else\originalTeX\fi
```

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 check for the existence of `\l@swedish` to see whether we have to do something here.

```
29.11 \ifx\undefined\l@swedish
29.12     \@nopatterns{Swedish}
29.13     \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.

[42] The file described in this section has version number ? and was last revised on ?. A contribution was made by Sten Hellman (`HELLMAN@CERNVM.CERN.CH`).

\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{Proof}%   <-- needs translation
  }%
```

\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}}
```

\swedishhyphenmins The swedish hyphenation patterns can be used with \lefthyphenmin set to 2 and \righthyphenmin set to 2.

```
\def\swedishhyphenmins{\tw@\tw@}
```

\extrasswedish The macro \extrasswedish performs all the extra definitions needed for the
\noextrasswedish Swedish language. The macro \noextrasswedish is used to cancel the actions of \extrasswedish. For the moment this macro is empty but it is defined for compatibility with the other language definition files.

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}
```

It is possible that a site might need to add some extra code to the babel macros. To enable this we load a local configuration file, swedish.cfg if it is found on TeX' search path.

```
\loadlocalcfg{swedish}
```

Our last action is to make a note that the commands we have just defined, will be executed by calling the macro `\selectlanguage` at the beginning of the document.

```
\main@language{swedish}
```

Finally, the category code of `@` is reset to its original value. The macrospace used by `\atcatcode` is freed.

```
\catcode`\@=\atcatcode \let\atcatcode\relax
```

⟨/code⟩

30 The Finnish language

The file `finnish.dtx`[43] defines all the language definition macros for the Finnish language.

For this language the character " is made active. In table 12 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 12: The extra definitions made by `finnish.ldf`

As this file needs to be read only once, we check whether it was read before. If it was, the command `\captionsfinnish` is already defined, so we can stop processing. If this command is undefined we proceed with the various definitions and first show the current version of this file.

```
30.1 ⟨*code⟩
30.2 \ifx\undefined\captionsfinnish
30.3 \else
30.4   \selectlanguage{finnish}
30.5   \expandafter\endinput
30.6 \fi
```

`\atcatcode` This file, `finnish.ldf`, may have been read while TEX is in the middle of processing a document, so we have to make sure the category code of `@` is 'letter' while this file is being read. We save the category code of the @-sign in `\atcatcode` and make it 'letter'. Later the category code can be restored to whatever it was before.

```
30.7 \chardef\atcatcode=\catcode‘\@
30.8 \catcode‘\@=11\relax
```

Now we determine whether the the common macros from the file `babel.def` need to be read. We can be in one of two situations: either another language option has been read earlier on, in which case that other option has already read `babel.def`, or `finnish` is the first language option to be processed. In that case we need to read `babel.def` right here before we continue.

```
30.9 \ifx\undefined\babel@core@loaded\input babel.def\relax\fi
```

[43] The file described in this section has version number ? and was last revised on ?. A contribution was made by Mikko KANERVA (`KANERVA@CERNVM`) and Keranen Reino (`KERANEN@CERNVM`).

Another check that has to be made, is if another language definition file has been read already. In that case its definitions have been activated. This might interfere with definitions this file tries to make. Therefore we make sure that we cancel any special definitions. This can be done by checking the existence of the macro `\originalTeX`. If it exists we simply execute it, otherwise it is `\let` to `\empty`.

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

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.

```
\ifx\undefined\l@finnish
    \@nopatterns{Finnish}
    \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.

```
\addto\captionsfinnish{%
  \def\prefacename{Esipuhe}%
  \def\refname{Viitteet}%
  \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{Proof}%  <-- needs translation!
  }%
```

`\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 make sure 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}}
\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 behavew a little different from `\-`.

```
\declare@shorthand{finnish}{"-}{\allowhyphens-\allowhyphens}
\declare@shorthand{finnish}{""}{\hskip\z@skip}
\declare@shorthand{finnish}{"=}{\hbox{-}\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\-{\allowhyphens
                           \discretionary{-}{}{}\allowhyphens}}
```

\finishhyphenmins The finnish hyphenation patterns can be used with \lefthyphenmin set to 2 and \righthyphenmin set to 2.

```
\def\finnishhyphenmins{\tw@\tw@}
```

It is possible that a site might need to add some extra code to the babel macros. To enable this we load a local configuration file, finnish.cfg if it is found on TEX' search path.

```
\loadlocalcfg{finnish}
```

Our last action is to make a note that the commands we have just defined, will be executed by calling the macro \selectlanguage at the beginning of the document.

```
\main@language{finnish}
```

Finally, the category code of @ is reset to its original value. The macrospace used by \atcatcode is freed.

```
\catcode`\@=\atcatcode \let\atcatcode\relax
⟨/code⟩
```

31 The Hungarian language

The file option `magyar.dtx`[44] defines all the language definition macros for the Hungarian language.

`\ontoday` For this language currently the only special definition that is added is the `\ontoday` command which works like `\today` but produces a slightly different date format used in expressions suh as 'on february 10th'.

As this file needs to be read only once, we check whether it was read before. If it was, the command `\captionsmagyar` is already defined, so we can stop processing. If this command is undefined we proceed with the various definitions and first show the current version of this file.

```
31.1 ⟨*code⟩
31.2 \ifx\undefined\captionsmagyar
31.3 \else
31.4   \selectlanguage{magyar}
31.5   \expandafter\endinput
31.6 \fi
```

`\atcatcode` This file, `magyar.ldf`, may have been read while TEX is in the middle of processing a document, so we have to make sure the category code of `@` is 'letter' while this file is being read. We save the category code of the @-sign in `\atcatcode` and make it 'letter'. Later the category code can be restored to whatever it was before.

```
31.7 \chardef\atcatcode=\catcode`\@
31.8 \catcode`\@=11\relax
```

Now we determine whether the the common macros from the file `babel.def` need to be read. We can be in one of two situations: either another language option has been read earlier on, in which case that other option has already read `babel.def`, or `magyar` is the first language option to be processed. In that case we need to read `babel.def` right here before we continue.

```
31.9 \ifx\undefined\babel@core@loaded\input babel.def\relax\fi
```

Another check that has to be made, is if another language definition file has been read already. In that case its definitions have been activated. This might interfere with definitions this file tries to make. Therefore we make sure that we cancel any special definitions. This can be done by checking the existence of the macro `\originalTeX`. If it exists we simply execute it, otherwise it is `\let` to `\empty`.

```
31.10 \ifx\undefined\originalTeX \let\originalTeX\empty \else\originalTeX\fi
```

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` to see whether we have to do something here.

```
31.11 \ifx\undefined\l@magyar
31.12     \@nopatterns{Magyar}
31.13     \adddialect\l@magyar0\fi
```

[44] The file described in this section has version number ? and was last revised on ?. A contribution was made by Attila Koppanyi (`attila@cernvm.cern.ch`). Later updates and suggestions by Árpád Bíró (`JZP1104@HUSZEG11.bitnet`), Istvan Hamecz (`hami@ursus.bke.hu`) and Horvath Dezso (`horvath@pisa.infn.it`).

An additional note about formatting Hungarian texts: One should invert the order of the number and text in things like chapter headings, page references etc. So one should write 'I. rész' instead of 'Part I', or '3. oldal' for 'page 3'.

For chapter headings this could be accomplished by a redefinition of the macros `\@makechapterhead` and `\@makeschapterhead`, for other instances this a lot harder to accomplish. Therefore I think complete document classes should be written to accomadate the needed formatting.

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 documentclasses provided with LaTeX.

```
\addto\captionsmagyar{%
  \def\prefacename{El\H osz\'o}%
  \def\refname{Referenci\'ak}%
  \def\abstractname{Kivonat}%
  \def\bibname{Bibliogr\'afia}%
  \def\chaptername{fejezet}%
  \def\appendixname{f\"uggel\'ek}%
  \def\contentsname{Tartalom}%
  \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}%
  \def\proofname{Proof}%   <-- needs translation
  }%
```

`\datemagyar` The macro `\datemagyar` redefines the command `\today` to produce Hungarian dates.

```
\def\datemagyar{%
  \def\today{\number\year.~\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.\or  2.\or  3.\or  4.\or  5.\or
    6.\or  7.\or  8.\or  9.\or 10.\or
   11.\or 12.\or 13.\or 14.\or 15.\or
   16.\or 17.\or 18.\or 19.\or 20.\or
   21.\or 22.\or 23.\or 24.\or 25.\or
   26.\or 27.\or 28.\or 29.\or 30.\or
   31.\fi}}
```

\ondatemagyar The macro \ondatemagyar produces Hungarian dates which have the meaning '*on this day*'. It does not redefine the command \today.

```
\def\ondatemagyar{%
  \number\year.~\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 The macro \extrasmagyar will perform all the extra definitions needed for the
\noextrasmagyar Hungarian language. The macro \noextrasmagyar is used to cancel the actions of \extrasmagyar. For the moment these macros are nearly empty; only the user command \ontoday to access \ondatemagyar is defined.

```
\addto\extrasmagyar{\let\ontoday\ondatemagyar}
\addto\noextrasmagyar{\let\ontoday\undefined}
```

It is possible that a site might need to add some extra code to the babel macros. To enable this we load a local configuration file, `magyar.cfg` if it is found on TEX' search path.

```
\loadlocalcfg{magyar}
```

Our last action is to make a note that the commands we have just defined, will be executed by calling the macro \selectlanguage at the beginning of the document.

```
\main@language{magyar}
```

Finally, the category code of @ is reset to its original value. The macrospace used by \atcatcode is freed.

```
\catcode`\@=\atcatcode \let\atcatcode\relax
```

⟨/code⟩

32 The Estonian language

The file `estonian.dtx`[45] 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 13. 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 13: 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`.

32.1 Implementation

Check whether the file has been read already.

32.1 ⟨∗code⟩
32.2 `\ifx\undefined\captionsestonian`

[45] The file described in this section has version number ? and was last revised on ?. The original author is Enn Saar, (`saar@aai.ee`).

```
\else
  \selectlanguage{estonian}
  \expandafter\endinput
\fi
```

Change the category code of `@`, as usual.

```
\chardef\atcatcode=\catcode`\@
\catcode`\@=11\relax
```

Check whether we need the common macros from the file `babel.def`.

```
\ifx\undefined\babel@core@loaded\input babel.def\relax\fi
```

We execute the macro `\originalTeX` to get rid of side effects that could be caused by language options used before.

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

If Estonian is not included in the format file (does not have hyphenation patterns), we shall use English hyphenation.

```
\ifx\undefined\l@estonian
  \@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{Proof}%   <-- 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}
```

If we are using the T1 output encoding, we should allow for the T1 input encoding, too (this has been chosen as the preliminary archiving standard by TWGML).

```
\edef\next{T1}
\ifx\f@encoding\next
   \addto\extrasestonian{%
      \catcode245=11 \catcode228=11 \catcode246=11 \catcode252=11
      \catcode178=11 \catcode186=11 \catcode213=11 \catcode196=11
      \catcode214=11 \catcode220=11 \catcode146=11 \catcode154=11
      \lccode245=245 \lccode228=228 \lccode246=246 \lccode252=252
      \lccode178=178 \lccode186=186 \lccode213=245 \lccode196=228
      \lccode214=246 \lccode220=252 \lccode146=178 \lccode154=186
      \uccode245=213 \uccode228=196 \uccode246=214 \uccode252=220
      \uccode178=146 \uccode186=154 \uccode213=213 \uccode196=196
      \uccode214=214 \uccode220=220 \uccode146=146 \uccode154=154
      \sfcode245=1000 \sfcode228=1000 \sfcode246=1000 \sfcode252=1000
      \sfcode178=1000 \sfcode186=1000 \sfcode213=999 \sfcode196=999
      \sfcode214=999 \sfcode220=999 \sfcode146=999 \sfcode154=999}
\fi
```

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.

```
\def\estonianhyphenmins{\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.

```
\edef\next{T1}
\ifx\f@encoding\next
  \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}{\textormath{\"{a}}{\ddot a}}
\declare@shorthand{estonian}{\textormath{\"{A}}{\ddot A}}
\declare@shorthand{estonian}{\textormath{\"{o}}{\ddot o}}
\declare@shorthand{estonian}{\textormath{\"{O}}{\ddot O}}
\declare@shorthand{estonian}{\textormath{\"{u}}{\ddot u}}
\declare@shorthand{estonian}{\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}{"-}{\allowhyphens\-\allowhyphens}
\declare@shorthand{estonian}{"|}{%
  \textormath{\penalty\@M\discretionary{-}{}{\kern.03em}%
              \allowhyphens}{}}
\declare@shorthand{estonian}{""}{\dq}
\declare@shorthand{estonian}{~~}{\til}
```

It is possible that a site might need to add some extra code to the babel macros. To enable this we load a local configuration file, `estonian.cfg` if it is found on TEX' search path.

```
\loadlocalcfg{estonian}
```

Select, finally, Estonian, and restore the category code of `@`. We are done.

```
\main@language{estonian}
\catcode`\@=\atcatcode \let\atcatcode\relax
```

⟨/code⟩

33 The Croatian language

The file `croatian.dtx`[46] defines all the language definition macros for the Croatian language.

For this language currently no special definitions are needed or available.

As this file needs to be read only once, we check whether it was read before. If it was, the command `\captionscroatian` is already defined, so we can stop processing. If this command is undefined we proceed with the various definitions and first show the current version of this file.

```
33.1 ⟨*code⟩
33.2 \ifx\undefined\captionscroatian
33.3 \else
33.4   \selectlanguage{croatian}
33.5   \expandafter\endinput
33.6 \fi
```

`\atcatcode` This file, `croatian.sty`, may have been read while TEX is in the middle of processing a document, so we have to make sure the category code of `@` is 'letter' while this file is being read. We save the category code of the @-sign in `\atcatcode` and make it 'letter'. Later the category code can be restored to whatever it was before.

```
33.7 \chardef\atcatcode=\catcode‘\@
33.8 \catcode‘\@=11\relax
```

Now we determine whether the the common macros from the file `babel.def` need to be read. We can be in one of two situations: either another language option has been read earlier on, in which case that other option has already read `babel.def`, or `croatian` is the first language option to be processed. In that case we need to read `babel.def` right here before we continue.

```
33.9 \ifx\undefined\babel@core@loaded\input babel.def\relax\fi
```

Another check that has to be made, is if another language definition file has been read already. In that case its definitions have been activated. This might interfere with definitions this file tries to make. Therefore we make sure that we cancel any special definitions. This can be done by checking the existence of the macro `\originalTeX`. If it exists we simply execute it, otherwise it is `\let` to `\empty`.

```
33.10 \ifx\undefined\originalTeX \let\originalTeX\empty \else\originalTeX\fi
```

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.

```
33.11 \ifx\undefined\l@croatian
33.12     \@nopatterns{Croatian}
33.13     \adddialect\l@croatian0\fi
```

The next step consists of defining commands to switch to (and from) the Croatian language.

[46] The file described in this section has version number ? and was last revised on ?. A contribution was made by Alan Paić (`paica@cernvm.cern.ch`).

\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{Glava}%
  \def\appendixname{Dodatak}%
  \def\contentsname{Sadr\v{z}aj}%
  \def\listfigurename{Slike}%
  \def\listtablename{Tablice}%
  \def\indexname{Indeks}%
  \def\figurename{Slika}%
  \def\tablename{Tablica}%
  \def\partname{Dio}%
  \def\enclname{Prilozi}%
  \def\ccname{Kopije}%
  \def\headtoname{Prima}%
  \def\pagename{Strana}%
  \def\seename{Vidi}%
  \def\alsoname{Vidi tako\dj er}%
  \def\proofname{Proof}%  <-- needs translation
  }%
```

\datecroatian The macro \datecroatian redefines the command \today to produce Croatian dates.

```
\def\datecroatian{%
\def\today{\number\day .~\ifcase\month\or
  sije\v{c}anj\or velja\v{c}a\or o\v{z}ujak\or travanj\or svibanj\or
  lipanj\or srpanj\or kolovoz\or rujan\or listopad\or studeni\or
  prosinac\fi
  \space \number\year}}
```

\extrascroatian The macro \extrascroatian will perform all the extra definitions needed for the
\noextrascroatian 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{}
```

It is possible that a site might need to add some extra code to the babel macros. To enable this we load a local configuration file, croatian.cfg if it is found on TeX' search path.

```
\loadlocalcfg{croatian}
```

Our last action is to make a note that the commands we have just defined, will be executed by calling the macro \selectlanguage at the beginning of the document.

```
\main@language{croatian}
```

Finally, the category code of @ is reset to its original value. The macrospace used by \atcatcode is freed.

```
\catcode`\@=\atcatcode \let\atcatcode\relax
```
⟨/code⟩

34 The Czech language

The file `czech.dtx`[47] 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`.

As this file needs to be read only once, we check whether it was read before. If it was, the command `\captionsczech` is already defined, so we can stop processing. If this command is undefined we proceed with the various definitions and first show the current version of this file.

```
34.1 ⟨*code⟩
34.2 \ifx\undefined\captionsczech
34.3 \else
34.4   \selectlanguage{czech}
34.5   \expandafter\endinput
34.6 \fi
```

`\atcatcode` This file, `czech.sty`, may have been read while TEX is in the middle of processing a document, so we have to make sure the category code of `@` is 'letter' while this file is being read. We save the category code of the @-sign in `\atcatcode` and make it 'letter'. Later the category code can be restored to whatever it was before.

```
34.7 \chardef\atcatcode=\catcode`\@
34.8 \catcode`\@=11\relax
```

Now we determine whether the the common macros from the file `babel.def` need to be read. We can be in one of two situations: either another language option has been read earlier on, in which case that other option has already read `babel.def`, or `czech` is the first language option to be processed. In that case we need to read `babel.def` right here before we continue.

```
34.9 \ifx\undefined\babel@core@loaded\input babel.def\relax\fi
```

Another check that has to be made, is if another language definition file has been read already. In that case its definitions have been activated. This might interfere with definitions this file tries to make. Therefore we make sure that we cancel any special definitions. This can be done by checking the existence of the macro `\originalTeX`. If it exists we simply execute it, otherwise it is `\let` to `\empty`.

```
34.10 \ifx\undefined\originalTeX \let\originalTeX\empty \else\originalTeX\fi
```

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.

```
34.11 \ifx\undefined\l@czech
34.12     \@nopatterns{Czech}
34.13     \adddialect\l@czech0\fi
```

[47] The file described in this section has version number ? and was last revised on ?. Contributions were made by Milos Lokajicek (`LOKAJICK@CERNVM`).

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 documentlasses 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{cc}%
  \def\headtoname{Komu}%
  \def\pagename{Strana}%
  \def\seename{viz}%
  \def\alsoname{viz tek\'e}%
  \def\proofname{Proof}%   <-- needs translation
  }%
```

\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 The macro \extrasczech will perform all the extra definitions needed for the
\noextrasczech 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 (ŏ). 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}{%
```

```
34.48     t\kern-.23em\raise.24ex\hbox{'}}
34.49   \DeclareTextCompositeCommand{\v}{OT1}{d}{%
34.50     d\kern-.13em\raise.24ex\hbox{'}}
34.51   \DeclareTextCompositeCommand{\v}{OT1}{l}{\lcaron{}}
34.52   \DeclareTextCompositeCommand{\v}{OT1}{L}{\Lcaron{}}}
```

`\lcaron` `\Lcaron` Fot the letters `l` and `L` we want to disinguish between normal fonts and monospaced fonts.

```
34.53 \def\lcaron{%
34.54   \setbox0\hbox{M}\setbox\tw@\hbox{i}%
34.55   \ifdim\wd0>\wd\tw@\relax
34.56     l\kern-.13em\raise.24ex\hbox{'}\kern-.11em%
34.57   \else
34.58     l\raise.45ex\hbox to\z@{\kern-.35em '\hss}%
34.59   \fi}
34.60 \def\Lcaron{%
34.61   \setbox0\hbox{M}\setbox\tw@\hbox{i}%
34.62   \ifdim\wd0>\wd\tw@\relax
34.63     L\raise.24ex\hbox to\z@{\kern-.28em'\hss}%
34.64   \else
34.65     L\raise.45ex\hbox to\z@{\kern-.40em '\hss}%
34.66   \fi}
```

It is possible that a site might need to add some extra code to the babel macros. To enable this we load a local configuration file, `czech.cfg` if it is found on TEX' search path.

```
34.67 \loadlocalcfg{czech}
```

Our last action is to make a note that the commands we have just defined, will be executed by calling the macro `\selectlanguage` at the beginning of the document.

```
34.68 \main@language{czech}
```

Finally, the category code of `@` is reset to its original value. The macrospace used by `\atcatcode` is freed.

```
34.69 \catcode`\@=\atcatcode \let\atcatcode\relax
34.70 ⟨/code⟩
```

35 The Polish language

The file `polish.dtx`[48] defines all the language-specific macros for the Polish language.

For this language the character `"` is made active. In table 14 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 14: The extra definitions made by `polish.sty`

As this file needs to be read only once, we check whether it was read before. If it was, the command `\captionspolish` is already defined, so we can stop processing. If this command is undefined we proceed with the various definitions and first show the current version of this file.

```
\ifx\undefined\captionspolish
\else
  \selectlanguage{polish}
  \expandafter\endinput
\fi
```

`\atcatcode` This file, `polish.sty`, may have been read while TEX is in the middle of processing a document, so we have to make sure the category code of `@` is ‘letter’ while this file is being read. We save the category code of the @-sign in `\atcatcode` and make it ‘letter’. Later the category code can be restored to whatever it was before.

```
⟨*code⟩
\chardef\atcatcode=\catcode‘\@
\catcode‘\@=11\relax
```

[48] The file described in this section has version number ? and was last revised on ?.

Now we determine whether the the common macros from the file `babel.def` need to be read. We can be in one of two situations: either another language option has been read earlier on, in which case that other option has already read `babel.def`, or `polish` is the first language option to be processed. In that case we need to read `babel.def` right here before we continue.

```
\ifx\undefined\babel@core@loaded\input babel.def\relax\fi
```

Another check that has to be made, is if another language definition file has been read already. In that case its definitions have been activated. This might interfere with definitions this file tries to make. Therefore we make sure that we cancel any special definitions. This can be done by checking the existence of the macro `\originalTeX`. If it exists we simply execute it.

```
\ifx\undefined\originalTeX
  \let\originalTeX\empty
\fi
\originalTeX
```

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.

```
\ifx\undefined\l@polish
  \@nopatterns{Polish}
  \adddialect\l@polish0\fi
```

The next step consists of defining commands to switch to (and from) the Polish language.

`\captionspolish` The macro `\captionspolish` defines all strings used in the four standard documentclasses provided with LaTeX.

```
\addto\captionspolish{%
  \def\prefacename{Przedmowa}%
  \def\refname{Bibliografia}%
  \def\abstractname{Streszczenie}%
  \def\bibname{Literatura}%
  \def\chaptername{Rozdzia\l}%
  \def\appendixname{Dodatek}%
  \def\contentsname{Spis rzeczy}%
  \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{Proof}%   <-- 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{"}}
%\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 14.

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
  \kern\pl@right\lower\pl@down\box1\kern-\pl@left #1}
```

\aob

The ogonek is placed with the letters 'a', 'A', 'e', and 'E'.

\Aob
\eob
\Eob

```
\def\aob{\sob a{.66}{.20}{0}{.90}}
\def\Aob{\sob A{.80}{.50}{0}{.90}}
\def\eob{\sob e{.50}{.35}{0}{.93}}
\def\Eob{\sob E{.60}{.35}{0}{.90}}
```

\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
  \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
  \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_ε.

```
  \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}{"r}{\textormath{\zkb}{\ddot r}}
\declare@shorthand{polish}{"R}{\textormath{\Zkb}{\ddot R}}
\declare@shorthand{polish}{"s}{\textormath{\'s}{\acute s}}
\declare@shorthand{polish}{"S}{\textormath{\'S}{\acute S}}
\declare@shorthand{polish}{"z}{\textormath{\'z}{\acute z}}
\declare@shorthand{polish}{"Z}{\textormath{\'Z}{\acute Z}}
```

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{\textquotedblleft{}}{\mbox{\textquotedblleft}}}
\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}{"-}{\allowhyphens-\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{\bbl@activate{"}}
\def\mdqoff{\bbl@deactivate{"}}
```

It is possible that a site might need to add some extra code to the babel macros. To enable this we load a local configuration file, `polish.cfg` if it is found on TEX' search path.

```
\loadlocalcfg{polish}
```

Our last action is to make a note that activate the commands we have just defined, will be executed by calling the macro `\selectlanguage` at the beginning of the document.

```
\main@language{polish}
```

Finally, the category code of `@` is reset to its original value. The macrospace used by `\atcatcode` is freed.

```
\catcode`\@=\atcatcode \let\atcatcode\relax
```

⟨/code⟩

36 The Slovak language

The file `slovak.dtx`[49] defines all the language-specific macros for the Slovak language.

For this language the macro `\q` is defined. It 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 ť.

As this file needs to be read only once, we check whether it was read before. If it was, the command `\captionsslovak` is already defined, so we can stop processing. If this command is undefined we proceed with the various definitions and first show the current version of this file.

```
36.1 ⟨*code⟩
36.2 \ifx\undefined\captionsslovak
36.3 \else
36.4   \selectlanguage{slovak}
36.5   \expandafter\endinput
36.6 \fi
```

`\atcatcode` This file, `slovak.sty`, may have been read while TEX is in the middle of processing a document, so we have to make sure the category code of `@` is 'letter' while this file is being read. We save the category code of the @-sign in `\atcatcode` and make it 'letter'. Later the category code can be restored to whatever it was before.

```
36.7 \chardef\atcatcode=\catcode‘\@
36.8 \catcode‘\@=11\relax
```

Now we determine whether the the common macros from the file `babel.def` need to be read. We can be in one of two situations: either another language option has been read earlier on, in which case that other option has already read `babel.def`, or `slovak` is the first language option to be processed. In that case we need to read `babel.def` right here before we continue.

```
36.9 \ifx\undefined\babel@core@loaded\input babel.def\relax\fi
```

Another check that has to be made, is if another language definition file has been read already. In that case its definitions have been activated. This might interfere with definitions this file tries to make. Therefore we make sure that we cancel any special definitions. This can be done by checking the existence of the macro `\originalTeX`. If it exists we simply execute it, otherwise it is `\let` to `\empty`.

```
36.10 \ifx\undefined\originalTeX \let\originalTeX\empty \else\originalTeX\fi
```

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.

```
36.11 \ifx\undefined\l@slovak
36.12     \@nopatterns{Slovak}
36.13     \adddialect\l@slovak0\fi
```

The next step consists of defining commands to switch to (and from) the Slovak language.

[49] The file described in this section has version number ? and was last revised on ?. It was written by Jana Chlebikova (`chlebik@euromath.dk`).

\captionsslovak The macro \captionsslovak defines all strings used in the four standard documentclasses provided with LaTeX.

```
\addto\captionsslovak{%
  \def\prefacename{\'Uvod}%
  \def\refname{Referencia}%
  \def\abstractname{Abstrakt}%
  \def\bibname{Literat\'ura}%
  \def\chaptername{Kapitola}%
  \def\appendixname{Dodatok}%
  \def\contentsname{Obsah}%
  \def\listfigurename{Zoznam obr\'azkov}%
  \def\listtablename{Zoznam tabuliek}%
  \def\indexname{Index}%
  \def\figurename{Obr\'azok}%
  \def\tablename{Tabu\q lka}%%% special letter l with hook
  \def\partname{\v{C}as\q t}%%% special letter t with hook
  \def\enclname{Pr\'{\i}loha}%
  \def\ccname{CC}%
  \def\headtoname{Komu}%
  \def\pagename{Strana}%
  \def\seename{vi\q d}%%%  Special letter d with hook
  \def\alsoname{vi\q d tie\v z}%%%  Special letter d with hook
  \def\proofname{Proof}%  <-- 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 augusat\or septembra\or okt\'obra\or
  novembra\or decembra\fi
    \space \number\year}}
```

\extrasslovak The macro \extrasslovak will perform all the extra definitions needed for the
\noextrasslovak Slovak language. The macro \noextrasslovak is used to cancel the actions of \extrasslovak. This currently means saving the meaning of one one-letter control sequence before defining it.

```
\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.

```
\def\slovakhyphenmins{\tw@\tw@}
```

It is possible that a site might need to add some extra code to the babel macros. To enable this we load a local configuration file, slovak.cfg if it is found on TeX' search path.

```
\loadlocalcfg{slovak}
```

Our last action is to make a note that the commands we have just defined, will be executed by calling the macro \selectlanguage at the beginning of the document.

```
\main@language{slovak}
```

Finally, the category code of `@` is reset to its original value. The macrospace used by `\atcatcode` is freed.

36.46 `\catcode‘\@=\atcatcode \let\atcatcode\relax`

36.47 ⟨/code⟩

37 The Slovanian language

The file `slovene.dtx`[50] defines all the language-specific macros for the Slovanian language.

For this language the character " is made active. In table 15 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 15: The extra definitions made by `slovene.ldf`

As this file needs to be read only once, we check whether it was read before. If it was, the command `\captionsslovene` is already defined, so we can stop processing. If this command is undefined we proceed with the various definitions and first show the current version of this file.

```
37.1 ⟨*code⟩
37.2 \ifx\undefined\captionsslovene
37.3 \else
37.4   \selectlanguage{slovene}
37.5   \expandafter\endinput
37.6 \fi
```

`\atcatcode` This file, `slovene.ldf`, may have been read while TEX is in the middle of processing a document, so we have to make sure the category code of `@` is ‘letter’ while this file is being read. We save the category code of the @-sign in `\atcatcode` and make it ‘letter’. Later the category code can be restored to whatever it was before.

```
37.7 \chardef\atcatcode=\catcode‘\@
37.8 \catcode‘\@=11\relax
```

Now we determine whether the the common macros from the file `babel.def` need to be read. We can be in one of two situations: either another language option has been read earlier on, in which case that other option has already read `babel.def`, or `slovene` is the first language option to be processed. In that case we need to read `babel.def` right here before we continue.

```
37.9 \ifx\undefined\babel@core@loaded\input babel.def\relax\fi
```

[50] The file described in this section has version number ? and was last revised on ?. Contributions were made by Danilo Zavrtanik, University of Ljubljana (YU) and Leon Žlajpah (`leon.zlajpah@ijs.si`).

Another check that has to be made, is if another language definition file has been read already. In that case its definitions have been activated. This might interfere with definitions this file tries to make. Therefore we make sure that we cancel any special definitions. This can be done by checking the existence of the macro `\originalTeX`. If it exists we simply execute it, otherwise it is `\let` to `\empty`.

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

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\undefined\l@slovene
    \@nopatterns{Slovene}
    \adddialect\l@slovene0\fi
```

The next step consists of defining commands to switch to the Slovanian 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 documentlasses provided with LaTeX.

```
\addto\captionsslovene{%
  \def\prefacename{Predgovor}%
  \def\refname{Literatura}%
  \def\abstractname{Povzetek}%
  \def\bibname{Literatura}%
  \def\chaptername{Poglavje}%
  \def\appendixname{Dodatek}%
  \def\contentsname{Kazalo}%
  \def\listfigurename{Slike}%
  \def\listtablename{Tabele}%
  \def\indexname{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{Proof}%  <-- needs translation
  }%
```

`\dateslovene` The macro `\dateslovene` redefines the command `\today` to produce Slovanian 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 Slovanian 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 slovanian group of shorthands should be used.

```
\initiate@active@char{"}
\addto\extrasslovene{\languageshorthands{slovene}}
\addto\extrasslovene{\bbl@activate{"}}
%\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}{"L}{\textormath{\v L}{\check L}}
\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}{"-}{\allowhyphens-\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}}{}}
```

It is possible that a site might need to add some extra code to the babel macros. To enable this we load a local configuration file, slovene.cfg if it is found on TEX' search path.

```
\loadlocalcfg{slovene}
```

Our last action is to make a note that the commands we have just defined, will be executed by calling the macro \selectlanguage at the beginning of the document.

```
\main@language{slovene}
```

Finally, the category code of @ is reset to its original value. The macrospace used by \atcatcode is freed.

```
\catcode`\@=\atcatcode \let\atcatcode\relax
```

⟨/code⟩

38 The Lower Sorbian language

The file `lsorbian.dtx`[51] It defines all the language-specific macros for Lower Sorbian.

As this file needs to be read only once, we check whether it was read before. If it was, the command `\captionlsorbian` is already defined, so we can stop processing. If this command is undefined we proceed with the various definitions and first show the current version of this file.

```
38.1 ⟨*code⟩
38.2 \ifx\undefined\captionslsorbian
38.3 \else
38.4   \selectlanguage{lsorbian}
38.5   \expandafter\endinput
38.6 \fi
```

`\atcatcode` This file, `lsorbian.ldf`, may have been read while TEX is in the middle of processing a document, so we have to make sure the category code of `@` is 'letter' while this file is being read. We save the category code of the @-sign in `\atcatcode` and make it 'letter'. Later the category code can be restored to whatever it was before.

```
38.7 \chardef\atcatcode=\catcode‘\@
38.8 \catcode‘\@=11\relax
```

Now we determine whether the the common macros from the file `babel.def` need to be read. We can be in one of two situations: either another language option has been read earlier on, in which case that other option has already read `babel.def`, or `lsorbian` is the first language option to be processed. In that case we need to read `babel.def` right here before we continue.

```
38.9 \ifx\undefined\babel@core@loaded\input babel.def\relax\fi
```

Another check that has to be made, is if another language definition file has been read already. In that case its definitions have been activated. This might interfere with definitions this file tries to make. Therefore we make sure that we cancel any special definitions. This can be done by checking the existence of the macro `\originalTeX`. If it exists we simply execute it, otherwise it is `\let` to `\empty`.

```
38.10 \ifx\undefined\originalTeX \let\originalTeX\empty \else\originalTeX\fi
```

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.

```
38.11 \ifx\undefined\l@lsorbian
38.12     \@nopatterns{Lsorbian}
38.13     \adddialect\l@lsorbian\l@usorbian\fi
```

The next step consists of defining commands to switch to (and from) the Lower Sorbian language.

[51] The file described in this section has version number ? and was last revised on ?. It was written by Eduard Werner (`edi@kaihh.hanse.de`).

\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
  }%
```

\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.

```
38.57 \addto\extraslsorbian{}
38.58 \addto\noextraslsorbian{}
```

It is possible that a site might need to add some extra code to the babel macros. To enable this we load a local configuration file, lsorbian.cfg if it is found on TEX' search path.

```
38.59 \loadlocalcfg{lsorbian}
```

Our last action is to make a note that the commands we have just defined, will be executed by calling the macro \selectlanguage at the beginning of the document.

```
38.60 \main@language{lsorbian}
```

Finally, the category code of @ is reset to its original value. The macrospace used by \atcatcode is freed.

```
38.61 \catcode`\@=\atcatcode \let\atcatcode\relax
38.62 ⟨/code⟩
```

39 The Upper Sorbian language

The file `usorbian.dtx`[52] It defines all the language-specific macros for Upper Sorbian.

As this file needs to be read only once, we check whether it was read before. If it was, the command `\captionusorbian` is already defined, so we can stop processing. If this command is undefined we proceed with the various definitions and first show the current version of this file.

```
\ifx\undefined\captionsusorbian
\else
  \selectlanguage{usorbian}
  \expandafter\endinput
\fi
```

`\atcatcode` This file, `usorbian.sty`, may have been read while TEX is in the middle of processing a document, so we have to make sure the category code of `@` is 'letter' while this file is being read. We save the category code of the @-sign in `\atcatcode` and make it 'letter'. Later the category code can be restored to whatever it was before.

```
\chardef\atcatcode=\catcode`\@
\catcode`\@=11\relax
```

Now we determine whether the the common macros from the file `babel.def` need to be read. We can be in one of two situations: either another language option has been read earlier on, in which case that other option has already read `babel.def`, or `usorbian` is the first language option to be processed. In that case we need to read `babel.def` right here before we continue.

```
\ifx\undefined\babel@core@loaded\input babel.def\relax\fi
```

Another check that has to be made, is if another language definition file has been read already. In that case its definitions have been activated. This might interfere with definitions this file tries to make. Therefore we make sure that we cancel any special definitions. This can be done by checking the existence of the macro `\originalTeX`. If it exists we simply execute it, otherwise it is `\let` to `\empty`.

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

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\undefined\l@usorbian
    \@nopatterns{Usorbian}
    \adddialect\l@usorbian0\fi
```

The next step consists of defining commands to switch to (and from) the Upper Sorbian language.

[52] The file described in this section has version number ? and was last revised on ?. It was written by Eduard Werner (`edi@kaihh.hanse.de`).

\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
  }%
```

\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{"}}
%\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.

```
\def\usorbianhyphenmins{\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@shorthandusorbian{}{"‘}{%
  \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{\usorbian@dq@disc ck}{c}}
\declare@shorthand{usorbian}{"C}{%
  \textormath{\usorbian@dq@disc CK}{C}}
\declare@shorthand{usorbian}{"f}{%
  \textormath{\usorbian@dq@disc f{ff}}{f}}
\declare@shorthand{usorbian}{"F}{%
  \textormath{\usorbian@dq@disc F{FF}}{F}}
\declare@shorthand{usorbian}{"l}{%
  \textormath{\usorbian@dq@disc l{ll}}{l}}
\declare@shorthand{usorbian}{"L}{%
  \textormath{\usorbian@dq@disc L{LL}}{L}}
\declare@shorthand{usorbian}{"m}{%
  \textormath{\usorbian@dq@disc m{mm}}{m}}
\declare@shorthand{usorbian}{"M}{%
  \textormath{\usorbian@dq@disc M{MM}}{M}}
\declare@shorthand{usorbian}{"n}{%
  \textormath{\usorbian@dq@disc n{nn}}{n}}
\declare@shorthand{usorbian}{"N}{%
  \textormath{\usorbian@dq@disc N{NN}}{N}}
\declare@shorthand{usorbian}{"p}{%
  \textormath{\usorbian@dq@disc p{pp}}{p}}
\declare@shorthand{usorbian}{"P}{%
  \textormath{\usorbian@dq@disc P{PP}}{P}}
\declare@shorthand{usorbian}{"t}{%
  \textormath{\usorbian@dq@disc t{tt}}{t}}
\declare@shorthand{usorbian}{"T}{%
  \textormath{\usorbian@dq@disc T{TT}}{T}}
```

and some additional commands:

```
\declare@shorthand{usorbian}{"-}{\penalty\@M\-\allowhyphens}
\declare@shorthand{usorbian}{"|}{%
  \textormath{\penalty\@M\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{\bbl@activate{"}}
\def\mdqoff{\bbl@deactivate{"}}
\def\ck{\allowhyphens\discretionary{k-}{k}{ck}\allowhyphens}
```

It is possible that a site might need to add some extra code to the babel macros. To enable this we load a local configuration file, `usorbian.cfg` if it is found on TEX' search path.

```
\loadlocalcfg{usorbian}
```

Our last action is to make a note that the commands we have just defined, will be executed by calling the macro `\selectlanguage` at the beginning of the document.

```
\main@language{usorbian}
```

Finally, the category code of `@` is reset to its original value.

```
\catcode`\@=\atcatcode
```

40 The Turkish language

The file `turkish.dtx`[53] defines all the language definition macros for the Turkish language[54].

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.

As this file needs to be read only once, we check whether it was read before. If it was, the command `\captionsturkish` is already defined, so we can stop processing. If this command is undefined we proceed with the various definitions and first show the current version of this file.

```
40.1 ⟨*code⟩
40.2 \ifx\undefined\captionsturkish
40.3 \else
40.4   \selectlanguage{turkish}
40.5   \expandafter\endinput
40.6 \fi
```

`\atcatcode` This file, `turkish.sty`, may have been read while TEX is in the middle of processing a document, so we have to make sure the category code of `@` is 'letter' while this file is being read. We save the category code of the @-sign in `\atcatcode` and make it 'letter'. Later the category code can be restored to whatever it was before.

```
40.7 \chardef\atcatcode=\catcode`\@
40.8 \catcode`\@=11\relax
```

Now we determine whether the the common macros from the file `babel.def` need to be read. We can be in one of two situations: either another language option has been read earlier on, in which case that other option has already read `babel.def`, or `turkish` is the first language option to be processed. In that case we need to read `babel.def` right here before we continue.

```
40.9 \ifx\undefined\babel@core@loaded\input babel.def\relax\fi
```

Another check that has to be made, is if another language definition file has been read already. In that case its definitions have been activated. This might interfere with definitions this file tries to make. Therefore we make sure that we cancel any special definitions. This can be done by checking the existence of the macro `\originalTeX`. If it exists we simply execute it.

```
40.10 \ifx\undefined\originalTeX \let\originalTeX\empty\fi
40.11 \originalTeX
```

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.

```
40.12 \ifx\undefined\l@turkish
40.13   \@nopatterns{Turkish}
40.14   \adddialect\l@turkish0\fi
```

[53] The file described in this section has version number ? and was last revised on ?.

[54] Mustafa Burc, `z6001@rziris01.rrz.uni-hamburg.de` provided the code for this file. It is based on the work by Pierre Mackay

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{Preface}% <-- This needs translation!!
  \def\refname{Ba\c svurulan Kitaplar}%
  \def\abstractname{Konu}%
  \def\bibname{Bibliografi}%
  \def\chaptername{Anab\"ol\"um}%
  \def\appendixname{Appendix}%
  \def\contentsname{\.I\c cindekiler}%
  \def\listfigurename{\c Sekiller Listesi}%
  \def\listtablename{Tablolar\i{}n Listesi}%
  \def\indexname{\.Index}%
  \def\figurename{\c Sekiller}%
  \def\tablename{Tablo}%
  \def\partname{B\"ol\"um}%
  \def\enclname{Ekler}%
  \def\ccname{G\"onderen}%
  \def\headtoname{Al\i{}c\i}%
  \def\pagename{Sayfa}%
  \def\subjectname{To}% <-- This needs translation!!
  \def\seename{see}% <-- This needs translation!!
  \def\alsoname{see also}% <-- This needs translation!!
  \def\proofname{Proof}% <-- This 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 The macro \extrasturkish will perform all the extra definitions needed for the
\noextrasturkish 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@:@` `\turksih@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}
```

It is possible that a site might need to add some extra code to the babel macros. To enable this we load a local configuration file, `turkish.cfg` if it is found on TEX' search path.

```
\loadlocalcfg{turkish}
```

Our last action is to make a note that the commands we have just defined, will be executed by calling the macro `\selectlanguage` at the beginning of the document.

```
\main@language{turkish}
```

Finally, the category code of `@` is reset to its original value. The macrospace used by `\atcatcode` is freed.

40.88 `\catcode‘\@\atcatcode \let\atcatcode\relax`

40.89 ⟨/code⟩

41 The Bahasa language

The file `bahasa.dtx`[55] 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.

As this file needs to be read only once, we check whether it was read before. If it was, the command `\captionsbahasa` is already defined, so we can stop processing. If this command is undefined we proceed with the various definitions and first show the current version of this file.

```
41.1 ⟨*code⟩
41.2 \ifx\undefined\captionsbahasa
41.3 \else
41.4   \selectlanguage{bahasa}
41.5   \expandafter\endinput
41.6 \fi
```

`\atcatcode` This file, `bahasa.sty`, may have been read while TEX is in the middle of processing a document, so we have to make sure the category code of `@` is 'letter' while this file is being read. We save the category code of the @-sign in `\atcatcode` and make it 'letter'. Later the category code can be restored to whatever it was before.

```
41.7 \chardef\atcatcode=\catcode‘\@
41.8 \catcode‘\@=11\relax
```

Now we determine whether the common macros from the file `babel.def` need to be read. We can be in one of two situations: either another language option has been read earlier on, in which case that other option has already read `babel.def`, or `bahasa` is the first language option to be processed. In that case we need to read `babel.def` right here before we continue.

```
41.9 \ifx\undefined\babel@core@loaded\input babel.def\relax\fi
```

Another check that has to be made, is if another language definition file has been read already. In that case its definitions have been activated. This might interfere with definitions this file tries to make. Therefore we make sure that we cancel any special definitions. This can be done by checking the existence of the macro `\originalTeX`. If it exists we simply execute it, otherwise it is `\let` to `\empty`.

```
41.10 \ifx\undefined\originalTeX \let\originalTeX\empty\fi
41.11 \originalTeX
```

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.

```
41.12 \ifx\undefined\l@bahasa
41.13   \@nopatterns{Bahasa}
41.14   \adddialect\l@bahasa0\fi
```

The next step consists of defining commands to switch to (and from) the Bahasa language.

[55] The file described in this section has version number ? and was last revised on ?.

\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}%
% Glossary: Daftar Istilah
  \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
  \def\seename{lihat}%
  \def\alsoname{lihat juga}%
  \def\proofname{Proof}%  <-- needs translation
  }
```

\datebahasa The macro \datebahasa redefines the command \today to produce Bahasa dates.

```
\def\datebahasa{%
  \def\today{\number\day~\ifcase\month\or
    Januari\or Februari\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 The macro \extrasbahasa will perform all the extra definitions needed for the
\noextrasbahasa 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{}
```

It is possible that a site might need to add some extra code to the babel macros. To enable this we load a local configuration file, bahasa.cfg if it is found on TeX's search path.

```
\loadlocalcfg{bahasa}
```

Our last action is to make a note that the commands we have just defined, will be executed by calling the macro \selectlanguage at the beginning of the document.

```
\main@language{bahasa}
```

Finally, the category code of `@` is reset to its original value. The macrospace used by `\atcatcode` is freed.

41.51 `\catcode‘\@=\atcatcode \let\atcatcode\relax`

41.52 ⟨/code⟩

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

Change History