Abstract

magyar.ldf, the Hungarian Babel module, was rewritten in the autumn of 2003 to obey most of the Hungarian typographical rules. This article describes some implementation issues, \TeX macro programming hacks, and \LaTeX typesetting trickery used in magyar.ldf. All features of the new magyar.ldf are enumerated, but only those having an interesting implementation are presented in detail. Most of the tricks shown are useful for developing other language modules.

1 The Name of the Language

Usually a Babel language module has the English name of that language. For example, the German module is called germanb.ldf, and not deutsch.ldf. The Hungarian module is an exception to this rule, because it has the name magyar.ldf, in which “magyar” is the Hungarian adjective meaning “Hungarian”. A similar exception is portuges.ldf for Portuguese. The letter “a” in word magyar has to be pronounced as in blah, and the consonant “gy” is the same as “d” in due.

The name of a language that a Babel language module (.ldf file) defines is usually specified as an argument of \LdfInit in the file. Thus, if czech.ldf is renamed to foo.ldf, it will have to be loaded with \usepackage[foo]{babel}, but to activate it, \selectlanguage{czech} should be used. This is not the case with magyar.ldf, because it detects its loaded filename using the \CurrentOption macro set by the \ProcessOptions command called from babel.sty. So whatever magyar.ldf is renamed to, that name is the one to pass to \selectlanguage.

The only reason why someone may want to rename an .ldf file is to load two different versions in the same \LaTeX run. This is possible with magyar.ldf, but the user should be aware that the control sequences defined by the two copies will interact with each other in an unpredictable way. Experiments have shown that it is possible to load two copies of magyar.ldf with different load options (this is the so-called dual load):

\PassOptionsToPackage{frenchspacing=yes}{magyar.ldf}
\PassOptionsToPackage{frenchspacing=no}{hungarian.ldf}
\usepackage[hungarian,magyar]{babel}

Despite the name hungarian.ldf above, the file magyar.ldf gets loaded twice, because Babel translates the language name hungarian to file name magyar.ldf, and magyar.ldf expects options for \CurrentOption, which depends on the language name passed to \usepackage[...]{babel}. Since the dual load feature of magyar.ldf is experimental, most of the load options cannot be different in the two copies. So the safest way to load two copies is to replace the occurrences of the word magyar in the second copy with something else.

The latest magyar.ldf (version 1.5) is not part of standard Babel yet, but it is available as part of Magyar\LaTeX (see section 4.1). Most of the typographical rules it tries to obey and problems it addresses were proposed in [1].

2 What an .ldf File Contains

An .ldf file is a Babel language module, which contains specific macros for the given language. It is loaded by babel.sty in the document preamble, at the time babel.sty itself is loaded. The macros defined in foo.ldf take effect only after changing the language with \selectlanguage{foo}. The default language is the one specified last in the \usepackage[...]{babel} command.

Babel itself contains the standard versions of the .ldf files as tex/generic/babel/*.ldf. In Babel 3.7 there are 41 of them; most are smaller than 10 kB. The largest files are: the old magyar.ldf defining the Hungarian language (25 kB), frenchb.ldf
Implementation Tricks in the Hungarian Babel Module

defining the French language (23 kB), \texttt{spanish.ldf} (21 kB), \texttt{bulgarian.ldf} (13 kB), \texttt{ukraineb.ldf} defining the Ukrainian language (12 kB), \texttt{russianb.ldf} (12 kB) and \texttt{greek.ldf} (9 kB). The new version of \texttt{magyar.ldf} is much larger than any of these: it is 178 kB. The size implies much more functionality, including several features unique to this new \texttt{magyar.ldf}—they will be discussed later in this document. Let’s proceed first by dealing with features common in most .ldf files.

2.1 Selecting the Hyphenation Pattern Set

\texttt{foo.ldf} must define the control sequence \texttt{\@foo} to be a number (\texttt{\newcount, \chardef, etc.}) representing the hyphenation pattern set to be used for that language. The language selection macro \texttt{\selectlanguage{\@foo}} calls \texttt{\language=\@foo}, which activates the hyphenation patterns for the language \texttt{foo}.

The patterns were (presumably) defined with the \texttt{\patterns} primitive at the time init\TeX was called to generate the format file. The exact file name containing the \texttt{\patterns} command is specified in the file \texttt{language.dat}. If there is a line “\texttt{foot fthyphen.tex}” in \texttt{language.dat}, then \texttt{\language=}\texttt{\@foo} will activate \texttt{\patterns} found in \texttt{fthyphen.tex}. In that case, \texttt{foo.ldf} should contain a line \texttt{\let\@foo=\@foo}. But this line is omitted in most actual .ldf files, because the Babel language name and the hyphenation pattern set name is the same (\texttt{language.dat} would contain an entry starting with \texttt{foo} in our example). Note that the file \texttt{fthyphen.tex} is read by init\TeX, not \TeX, so the format files have to be re-generated each time \texttt{fthyphen.tex} is changed.

Three different hyphenation pattern sets have been proposed for the Hungarian language (namely, \texttt{huhyph3.tex, huhyphc.tex} and \texttt{huhyphf.tex}). All of them are maintained by Gyula Mayer [4]. The most important difference among them is the way they hyphenate at subword boundaries of compound words. The document author can select any of these three by providing the appropriate load option to \texttt{magyar.ldf} (discussed later). The options work by redefining \texttt{\@magyar} to be one of \texttt{\@magyar, \@magyars, \@magyarf, \@magyarh, \@magyarb, \@magyarw}.

There are two different correct ways to hyphenate compound words in Hungarian. \texttt{magyarf} hyphenates the most common foreign compound words of Hungarian text phonetically (e.g. \texttt{szink-ron}, meaning synchronous), while \texttt{magyarh} hyphenates them on the subword boundary (e.g. \texttt{szin-kron}). \texttt{magyar3} is the old version of the hyphenation patterns which hyphenates most composite words phonetically (even non-foreign ones), save only a few exceptions listed explicitly. However, in all the three cases, hyphenation of foreign words cannot be perfect, because all of them cannot be specified in \texttt{\patterns}.

\texttt{magyar.ldf} redefines \texttt{\@magyar} depending on the \texttt{hyphenation=} load option. If a given pattern set may be missing from the user’s system, \texttt{magyar.ldf} falls back to another set with a meaningful warning message. Hyphenation is disabled not by choosing \texttt{\language0}, as \texttt{Babel} does, because \texttt{\language0} may contain valid patterns for a different language, but rather \texttt{\language255}, which is very likely to be unused since \texttt{\@TeX} assigns \texttt{\language} numbers from zero.

2.2 Defining Captions

\texttt{\@TeX} generates some words and phrases automatically. For example, \texttt{\tableofcontents} should emit the phrase “Table of contents” in the native language. The same applies for \texttt{\caption}s of figures and tables, and also for \texttt{\chapter} titles. Thus \texttt{Babel} expects \texttt{foo.ldf} to define a macro called \texttt{\caption{foo} containing definitions like \texttt{\def\abstractname{\texttt{\@date}}} \texttt{\abstractname} \texttt{\@language255}. These definitions are executed by \texttt{\selectlanguage} each time the language is activated. So it is possible to have an English and then a Hungarian chapter in a book numbered ‘Chapter 1’ and ‘2. fejezet’, respectively:

\begin{verbatim}
\chapter{foo} ...
\selectlanguage{magyar}\chapter{bar}
\end{verbatim}

\texttt{magyar.ldf} has the proper definitions of Hungarian phrases. Some words contain accented letters, which are specified as commands (e.g. \texttt{\‘a} for \texttt{\‘a}) and not single 8-bit characters, so their interpretation does not depend on the active input encoding, i.e. the load option of \texttt{inputenc.sty}.

2.3 Generating Dates

\texttt{foo.ldf} should define a macro \texttt{\datefoo} to define the macro \texttt{\today}, which emits a date (specified by the \texttt{\year, \month, \day} registers) correctly for that language. The month name should be printed as a non-abbreviated word. The definition of \texttt{\today} is used by \texttt{\@date} invoked in \texttt{\maketitle} in the standard document classes.

In addition to defining \texttt{\today}, \texttt{magyar.ldf} defines the macro \texttt{\ontodatemagyar} to further define \texttt{\ontoday}, which emits the date with the Hungarian equivalent of English \texttt{on}. The Hungarian language has suffixes instead of prepositions, and each suffix has several forms which must follow the vowel harmony of the word it is suffixed to. Thus “on March
The macro to be saved for \DeclareTextCompositeCommand{\v}{OT1} is \OT1\v (with the second backslash being part of the control sequence), but assigning the new meaning would be problematic, since \DeclareTextCompositeCommand can be used only in the preamble. Thus the correct solution would involve fiddling with undocumented \LaTeX{} internals; which is probably why czech.ldf contains the problematic workaround using \AtBeginDocument.

Fortunately, the only non-English letters in the Hungarian language are accented vowels (á, é, í, ó, õ, õ, õ, õ and ü), which are all part of the T1 encoding. The letters õ and õ with the special Hungarian double-acute accent are missing from the Latin-1 encoding (ISO-8859-1), but are part of Latin-2. So authors dealing with Hungarian are encouraged to use \usepackage{latin2}{inputenc}.\footnote{The most common incorrect letters found in Hungarian texts are ő and ü: their presence is caused by software incapable of using Unicode or the Latin-2 encoding. These letters can be seen even on some huge advertisement banners on streets in Hungary. These texts were not typeset by \LaTeX{}, of course!} \usepackage{tienc} is also recommended, so \LaTeX{} will be able to hyphenate words containing accented letters.

The finest Hungarian books have accents lowered a little bit. This is accomplished for the dieresis accent (˝) by calling the \umlautlow command (defined by \texttt{Babel}) in \texttt{extrasmagyar}. No serious attempt is made to make this work for all three Hungarian accents, because the technology \umlautlow is based on works only for the OT1 encoding (which composes accented letters), but most Hungarian texts use the T1 encoding to allow hyphenation in words with accented letters.

The lowering of accents is possible using virtual fonts. But \LaTeX{} font families come with too many variations and design sizes, so the virtual font generation would need to be automated. The macro \lower@umlaut in babel.def lowers accents by forcing their top to be 1.45ex above the baseline. The \accent primitive lowers its accent by \fontdimen5\font=1ex, so the top of the accent can be forced to 1.45ex by setting \fontdimen5\font=\ht0 + 0.45ex, where \ht0 is the height of the accent character (\char127 in the OT1 encoding).

The lowering, in the case of ˝, is as small as 0.43558pt. Even this tiny displacement can make a visible difference: “ít < ii.” The lowering method could be made adaptive by rendering the glyphs involved at high resolution, measuring the number of pixels between the accent and the letter vertically, and then lowering the accent so the distance will be a prescribed constant value.

2.4 Minimum Hyphenation Length
\LaTeX{} won’t insert an implicit hyphen into the first \lefthyphenmin characters of words, nor in the last \righthyphenmin characters. The default \LaTeX{} values for these are 2 and 3, respectively, which are suitable for the English language. foo.ldf can override the default by defining the macro \foophyphenmins to be 1r, two digits specifying the left and the right minimum, respectively.

What magyar.ldf does depends on its load options. The default is to follow Hungarian typography: \def\magaryphyphenmins{22}.

Nine of the 41 .ldf files in Babel 3.7 do only the customizations described to this point. 25 languages go a little beyond these, and 7 languages go much beyond. Of those 25 + 7 languages that go beyond, we will compare frenchb.ldf in detail to magyar.ldf, because French and Hungarian share some typographical rules.

2.5 Defining Special Letters
Many languages have letters that are missing from the standard OT1 encoding, and some characters are missing even from T1. These should be implemented in .ldf files as control sequences. It is a common practice to modify the meaning of an existing letter, for example czech.ldf contains \DeclareTextCompositeCommand{\v}{OT1}{t}{...} . However, this declaration is contained in \AtBeginDocument, so they are in effect even when not the Czech language is active. This should have been avoided.

The correct solution is to use the extras facility provided by Babel: foo.ldf can have a macro \extrasfoo, which is executed each time the language foo is activated; and the macro \noextrasfoo is executed when the active language is about to change (because of a \selectlanguage command or when the end-of-group is reached). It is a common practice in \extrasfoo to save the meaning of a macro with \babel@save, or a meaning of a count, dimen or skip register with \babel@savevariable. The saved meanings will be restored just after \noextrasfoo is executed. Babel provides the command \addto that can append tokens to the definition of an existing macro. The idiom \addto{\extrasfoo}{\babel@save}{\bar}{\def\bar{\foo-bar}} is typical, which gives a new meaning to \bar while the language foo is active.

15" is emitted as március 15-én, but “on March 16” is március 16-án, showing that the -án/-én suffix has two forms.
Neither home users nor professionals use lowered accents in Hungary today, not even with books created with \LaTeX{} — the original fonts with the T1 encoding are acceptable enough not to bother changing. Typo\TeX{} Ltd., one of the biggest Hungarian publishing houses using \TeX{}, developed the OM fonts in the early 1990s for use with plain \TeX{}. The OM fonts are a variation of CM fonts with Hungarian accented glyphs added (with lowered accents). However, it is not worth creating .fd files for the OM fonts for use with \LaTeX{}, because with the same amount of work new virtual fonts could be created from the EC fonts, which would take advantage of the full T1 character set, and existing, hinted fonts in Type 1 formats (such as the CM-Super fonts).

### 2.6 Hyphenation of Long Double Consonants

Hyphenating long double consonants in Hungarian is a difficult typographical problem. For example, the correct way to hyphenate the double consonant \textit{tty} is \textit{ty} + \textit{ty} in the word \textit{hattyú} (“swan”) is \textit{haty-tyú}. (There is a similar problem in German with words containing \textit{ck}; [5] documents more languages with more exceptions.) The long double consonants involved are: \textit{ccs}, \textit{ddz}, \textit{ddzs}, \textit{gg}, \textit{ggy}, \textit{tty} and \textit{zzs}. \TeX{}’s automatic hyphenation algorithm cannot deal with such exceptions, but adding ligatures dealing with the dash inserted by the implicit hyphenation can solve the problem. The simple trick of having \texttt{\patterns{ttty ggy1y}} and \texttt{t+y+y} seems to solve the problem, because it hyphenates \textit{tty} as \textit{ty-ty}, but it also inserts an extra \textit{y} before the hyphen in \textit{fat-szalad}. Normal patterns will also insert an implicit hyphen into \textit{botcsinálta}, yielding \textit{bot-csinálta}. The ligature program above would then incorrectly alter that to \textit{ty-ty-ty} onto the character set, and existing, hinted fonts in Type 1 formats (such as the CM-Super fonts).

So a more elaborate set of ligatures would have to be constructed, to detect the context of the hyphen and insert the \textit{y} only into \textit{t-ty}, yielding \textit{t-y}. Or, equivalently, using \texttt{\patterns{ttty ggy1y}} with context-sensitive ligatures changing \texttt{tt1y} to \texttt{tt-y} and \texttt{gg-y} to \texttt{gy-gy}, etc. This solution uses up many character positions from the font, and many extra ligatures are involved. Also, the user must know that to produce an actual \textit{t-ty} (which almost never appears in Hungarian), \texttt{t()}-\texttt{ty} must be used.

All of this can be accomplished using virtual fonts. The author has tested to see that the concept works by compiling \texttt{aer10.vf} to \texttt{aer10.vpl} and modifying the (LIGTABLE). However, automation of the virtual font generation is work remaining for the future.

Hyphenation of the double two-character consonants \textit{ggy} and \textit{ssz} is similar to \textit{tty}. However, compound words such as \textit{leggyakoribb} (“most frequent”) and \textit{vasszekér} (“iron chariot”) should be hyphenated at the subword boundary without the addition of extra letters, i.e. as \textit{leg-gyakoribb} and \textit{vas-szekér}. Extra \texttt{\patterns{} may be added, for example \patterns{ggy1y .leg1g4yakoribb.}, to disable insertion of \textit{y} for each important compound word. This is quite straightforward, because it does not require more ligatures (apart from the context sensitive ligature program changing \textit{gg-y} to \textit{gg-gy}).

One might suggest context-sensitive ligatures could be avoided if \textit{ty} is introduced as a new single letter. But this won’t work because, step (1): ‘\textit{tty}’ has to be converted to ‘\textit{ty-ty}’ using more than one ligature, and then step (2): further conversion to ‘\textit{t-ty}’ if there is no line break, but \TeX{} won’t run its hyphenation algorithm in the middle of ligature processing, between steps (1) and (2).

The current approach of \texttt{magyar.ldf} for handling long double consonants is a compromise. By default, the patterns do not hyphenate those consonants, and the character ‘\texttt{t}’ is made active (with the standard \texttt{Babel} command \texttt{\declare@shorthand{}}, so that, for example ‘\texttt{tty} emits t\texttt{\nobreak \discretionary{y-}{}{}tty \nobreak \hskip \z@skip}. The first \texttt{\nobreak} is used to enable automatic hyphenation before the ‘\texttt{tty}’ construct, and the last \texttt{\nobreak} plus the \texttt{\hskip} enable hyphenation after the construct. The word typed as \texttt{megho’sszabbit} will be hyphenated as \texttt{meg-hosz-szab-bit}. Similar shorthands are added for the other long consonants. The compromise is that the user has to be aware that he has to insert ‘\texttt{t}’ manually. A Perl script named \texttt{ccs_extract.pl} was developed to collect all occurrences of double consonants in a document so the user can review them and decide about the insertion of ‘\texttt{t}’ for each.

### 2.7 Table and Figure Captions

The document class defines \texttt{\maketitle}, which is responsible for typesetting a caption for tables and figures. Some \texttt{.ldf} files, including \texttt{frenchb.ldf} and \texttt{magyar.ldf} override the default behaviour. \texttt{magyar.ldf} changes the colon separating the caption heading (e.g. “1. táblázat”, or “Table 1”) from the caption text to a full stop, in keeping with Hungarian typography. Furthermore, the \texttt{longcaption=} load option controls what should happen when the caption doesn’t fit in a single line: whether it should

---

\footnote{\texttt{(LABEL C t)} \texttt{(LIG C - C y)} in the \texttt{.pl} file.}
be centered, and whether there should be a line break after the caption heading.

The `tablecaptions=` and `figurecaptions=` load options control the appearance of the caption heading by redefining both \fnum\table and \fnum\figure. The default in both cases is to follow Hungarian typography, which requires the number to precede the table name.

2.8 Between the Section Title and the Section Number

The default definitions of \@ssect and \@sect separate the section number from the section title with a \quad. In Hungarian typography, only an \enskip is needed, and a dot has to be inserted after the number. The old version of magyar.ldf changed \@sect etc., but this caused conflicts with the AMS document classes and other packages, so that strategy has been given up in the new version of magyar.ldf. Instead, dots were moved into the \numberline, which adds the dot to \tableofcontents lines (being careful not to append the dot twice, see dot stripping code in section 2.22), and to \@seccntformat, which adds the dot and \enskip to the titles. The AMS document classes do not use \@seccntformat, so the AMS-specific \tocsection and \tocappendix commands had to be modified.

\numberline also adds a dot after table and figure numbers in the \listoftables and \listoffigures, but the dot is needed there, too, anyway.

All three TOCs share a common problem related to language changes. Each time the language is changed, Babel emits changing commands to the three TOC files, so when they are re-read, each line comes in its appropriate language. The implementation has a flaw, however, because the \write to the TOC files gets executed when the page is shipped out, and the order of \writes on the same page follows the document structure: \writes in top insertions precede and bottom insertions follow those in the main text. So when a table or figure is moved to the top of the page, the writing of its TOC entry, together with the \selectlanguage command emitted by Babel is moved away, so \selectlanguage commands in the TOC files are reordered, which is wrong. The solution is to emit a \selectlanguage command for each TOC entry, so the TOC entries can be freely reordered. magyar.ldf implements this solution as a local fix, but it should be fixed generally in a new version of Babel.

2.9 Spacing Around Punctuation

It is quite easy to add extra space after punctuation characters with \scode (see “space factor” in chapter 13 of [3]). The \LaTeX \nonfrenchspacing command (which is activated by default) assigns a space factor of 3000 to , , ? and !, 2000 to :, 1500 to ;, and 1250 to .

However, adding extra space before punctuation needs a different approach. Both frenchb.ldf and magyar.ldf make the characters :, ;, ! and ? active with the Babel \initiate@active@char interface, and insert unbreakable space in horizontal mode (\ifhmode) just before the punctuation character. This feature of magyar.ldf can be turned off using the \activespace= load option, partly because making these four common characters active may lead to incompatibility with other packages, and partly because the extra space before punctuation is very rare in current Hungarian documents. In French typography, about one-third of a normal space is required before punctuation, and if it is not possible to add that amount with the typesetting technology, one full space should be added. However, in Hungarian, the fallback strategy is to omit the extra space.

The last action the active punctuation character should do is insert itself, but typing it verbatim into the definition will lead to an infinite loop. For example, \catcode?=13 \def\kern.1em ? will loop infinitely. The solution is to use \string? in place of the last ?, so its catcode will be changed to 12 (other). Using \edef with this approach will make the macro a little bit faster, because \string will be executed only once, at load time.

2.10 Quoted Material

In English, text can be quoted using ‘single’ or “double” quotation marks. These can be nested into each other both ways. Hungarian provides three nesting levels: „outer” „middle” „inner” “en“. Although the guillemets symbols are missing from the CM fonts with OT1 encoding, this is not a serious problem, since Babel provides them (using the \ll relation: « and »), and Hungarian text should be typeset with the T1 font encoding anyway, to allow hyphenation of words with accented characters.

frenchb.ldf provides \LasyGuillemets and \CyrillicGuillemets so the user can select the origin of the replacement guillemets. magyar.ldf relies on the defaults provided by Babel in the hope that the T1 encoding is used, so replacements are not needed. magyar.ldf doesn’t adjust spacing around
Implementation Tricks in the Hungarian Babel Module

the quotation symbols, but provides a \textqq command which emits quotations with proper nesting and spacing. For example, \textqq{outer \textqq{middle} \textqq{inner}} gives the above three-level sample. \textqq does English quotations (with two alternating levels) when the Hungarian language is not active.

2.11 List Environments

The default spacing, indentation and label item generation of list environments (such as itemize and description) are incorrect for Hungarian. The languageattribute and labelitemsize load options control whether labels are modified to match Hungarian traditions. Five levels of depth are provided for both itemize and enumerate. The maximum depth is hardwired to the \texttt{tex} definitions of these environments, so the \texttt{ifnum@enumdepth}>3 test had to be changed to

\texttt{expandafter \texttt{ifx\csname \texttt{labelitemsize} \\\texttt{romannumeral}\the@enumdepth\endcsname\relax}

(and similarly for \texttt{@itemdepth}).

Although the vertical space that the standard document classes leave around lists is too large, and the indentation is also incorrect, these problems have not yet been solved in \texttt{magyar.ldf}. (frenchb.ldf modifies \texttt{itemsep} and other spacing dimensions to match French typographical rules.)

2.12 Modularity Using Load Options

The user can customize .ldfs using Babel's language attribute facility. For example, greek.ldf has \texttt{\declareattribute[greek] \{polutoniko\} {.}, and so if the user loads greek.ldf with

\texttt{\usepackage[greek]{babel}}

\texttt{\languageattribute[greek]{polutoniko}}

the code in the .. is run when \texttt{languageattribute} is called. If present, \texttt{languageattribute} must be part of the document preamble, and the .ldf file must be already loaded.

The fundamental problem with language attributes is that the user can pass only declared keywords, and not arbitrary data to the .ldf file, and — since attributes are processed too late — they cannot be used to control which parts of the .ldf files should be loaded.

Thus, magyar.ldf follows a different approach. Options are (key)=(value) pairs, which can be declared any time before \texttt{magyar.ldf} is loaded. The set of keys is fixed, but values can be arbitrary. It is the responsibility of the macro belonging to the key to verify that the syntax of the value is correct. None of the other .ldf files provide load option support this flexible. This option-passing scheme is similar to keyval.sty, but \texttt{magyar.ldf} doesn’t actually use keyval.sty, because of a general design policy to avoid dependencies.

Since .ldf file names are the \texttt{tex} options to \texttt{babel.sty} in the \texttt{\usepackage[\ldots]{babel}} line, it is not possible to pass options to the individual .ldf files directly. However, \texttt{tex} provides the command \texttt{\PassOptionsToPackage}, which declares options for a package before the package is loaded. So for example, \texttt{\PassOptionsToPackage{a=b,c=d}{foo.bar}} appends a=b,c=d to the macro \texttt{\opt@foo.bar}. \texttt{magyar.ldf} examines \texttt{\opt@magyar.ldf}, so for example passing options with \texttt{\PassOptionsToPackage{titles=\enskip}{magyar.ldf}} forces the space in section headings between the section number and the section title to be \texttt{\enskip}. (For compatibility reasons, \texttt{magyar.ldf} also processes the contents of \texttt{\magyarOptions} as options. This is useful to make \texttt{magyar.ldf} work with plain \texttt{tex}.)

\texttt{tex} macro wizards may enjoy studying the option parsing code in \texttt{magyar.ldf}. The entry point of the routine is \texttt{\processOptions}, whose argument is a comma-separated option list of (key)=(value) pairs. The routine calls \texttt{\processOption{key} \{value\} for each pair found. This code is shown in figure 1.

2.13 Default Option Sets

Since there are 51 load options in the current \texttt{magyar.ldf}, the user should not be forced to know all of them. Reasonable defaults are provided (namely, \texttt{defaults=over-1.4}), so novice users can simply proceed. Intermediate users can select one of the five defaults, and possibly change a few options they don’t like in their preferred default, and only expert users will change many options individually.

The number of bytes loaded were measured in a recent version of \texttt{magyar.ldf}, totalling 177 353 bytes. Out of that 177 kB, 32 417 bytes were used for initialization and providing the load option support framework, and declaring the options for the five defaults. After that, 138 872 bytes were used for implementing features selected by the load options. In the descriptions below, the number of feature bytes skipped is listed. (The larger the number, the less of \texttt{magyar.ldf} is processed at load time.)

The default sets are:

1. \texttt{over-1.4} (10 720 bytes not loaded) This is the default among the defaults. Its main goal is to make all documents with the previous version of \texttt{magyar.ldf} (1.4) compile with the new version and to provide emergency bugfixes to incompatibility problems caused by the old version. It introduces a few essential typographical changes which have little impact on line
2.14 Skipping Parts of the Input File

Since some parts of magyar.ldf can be disabled using load options, as we have seen, it is desirable to skip them completely. The easiest way of skipping part of TeX code is wrapping it into \ifnum\MyFeature<1 ...\fi. But this kind of skipping will consume hash memory for new control sequences skipped over, and it also requires that the skipped part is properly nested with respect to \if...s. magyar.ldf defines the following macro to do skipping without these flaws.

\def\skiplong#1{\fi
\bgrou% so `\) would close it
\catcode\string`13
\lccode\string`=\string`
\lowercase{\let`\fi}\
\catcode\string`14 % comment, save hash memory

\gobble\iftrue
\def\skiplong#1{\fi
\bgrou% so `\) would close it
\catcode\string`13
\lccode\string`=\string`
\lowercase{\let`\fi}\
\catcode\string`14 % comment, save hash memory

and page breaks; it disables big, eye-catching changes. It makes most new commands available, but doesn’t turn new features on.

2. =compat-1.4 (82 890 bytes not loaded) Implements a strict compatibility mode with version 1.4 of magyar.ldf. Documents look about the same (exact match not guaranteed), even when the output is typographically incorrect. It does not define new commands such as \told or \emitdate.

3. =safe$t (124 679 bytes not loaded) Turns off all features, reverts to L\TeX{} and \texttt{Babel} defaults in every respect. It is useful only for debugging purposes: if a document doesn’t compile, but it compiles with defaults=safest, individual options can be turned on one-by-one to see which is causing the compatibility problem.

4. =prettiest (1 221 bytes not loaded) Turns on all new features, and tries to follow Hungarian typography in the prettiest, most advanced way. It is possible that compatibility problems will arise with other packages, although the author is not currently aware of any.

5. =hu-min (1 317 bytes not loaded) Follows Hungarian typographical rules as closely as possible. Compliance is not complete, of course, because some aspects are not implemented; thus they are not covered by load options, and they cannot be controlled using defaults. If typographical rules allow choice (e.g. the first paragraph of a section may or may not be indented), the easiest and most compatible solution is chosen (e.g. accept the indentation defined by the document class).

Figure 1: Option parsing code. Comments: (1) ignores extra commas, detects them by testing whether #1 is empty; (2) this is a \texttt{\fi} when expanded, but doesn’t count as a \texttt{\fi} when being skipped over because its surrounding condition is false. The real \texttt{\fi} won’t be expanded, because it is parsed as the parameter terminator of \texttt{processOptions@b}. (3) needs \#1\#2 instead of just \#1, so Te\TeX{} will ignore space tokens in front of \#1. As a side effect, when the option begins with =, the = will be put into \#1, so \texttt{missingKey} can be reported. (4) There are four different cases in which \texttt{processOptions@b} can be invoked. The exact case is determined by how the macro parameter text separates parameters. The cases are: Normal case: \#1\#2 is the key, \#3 is the value, \#4 =,; MissingArg case =\texttt{(}=\texttt{value}) is missing, or \texttt{\{key\}} is missing, but \texttt{=\texttt{(}value\texttt{)}} is present; \#1\#2 is \texttt{\{key\}}, or \texttt{=\texttt{(}value\texttt{)}}, \#3 and \#4 are empty; Terminator case: \#1\#2 is \texttt{\{h fuzz\}}, \#3 and \#4 are empty; OnlySpace case: \#1 is ,; \#2, \#3 and \#4 are empty. (5) \texttt{\gobble} removes the call of \texttt{processOptions@a} at the end of the macro, so the iteration is finished.
Detecting Digits for Definite Articles

\ref, \pageref and \cite generate numbers, which are often prefixed by the definite article in Hungarian. The construct \texttt{the \ref{foo}}; works fine in English, but the Hungarian definite article has two forms: \texttt{a} and \texttt{az}. \texttt{Az} must be used if the following words (as pronounced) starts with a vowel, and \texttt{a} must be used for consonants. So we need a macro that generates the definite article for numbers automatically. \texttt{magyar.ldf} contains the macro \texttt{\az}, which prefixifies its argument by either \texttt{a} or \texttt{az}. This kind of macro is at present unique to \texttt{magyar.ldf}; other .ldf files apparently do not implement solutions for similar problems in other languages.

Unfortunately, \texttt{\az} is not expandable, because it redefines the meaning of several commands before processing its argument. \texttt{\az} works by half-expanding its argument (fully expanding, of course, \texttt{\ref}, \texttt{\pageref} and \texttt{\cite}), ignoring braces and most control sequences, non-digit and non-letter characters, changing \texttt{\romannumeral} to \texttt{\number} (so that \texttt{x} will become \texttt{az x}, but \texttt{\az{\romannumeral 10}} yields \texttt{a x}), looking for a number, a word or a single letter in the beginning (in fact, \texttt{az} has to be emitted if the starting digit is 5, and \texttt{a} has to be emitted if the starting digit is not 5 or 5). For words, single letters and positive numbers not beginning with \texttt{I}, the proper definite article depends on the first letter only.

For numbers starting with \texttt{I}, the definite article must be \texttt{az} if and only if the number of digits is \(3k + 1\) for an integer \(k\). For example, the Hungarian words for 1, 12, 123, 1000 are \texttt{egy}, \texttt{tizenkétő}, \texttt{százhuszonhárom}, \texttt{ezer}, respectively, and the definite forms are \texttt{az 1}, \texttt{a 12}, \texttt{a 123}, and \texttt{az 1000}, respectively. So we have to count the number of digits of a number.

It is not necessary to have \texttt{10 \ifnum} commands to test whether macro argument \texttt{#1} is a digit: it almost always works fine to use \texttt{\ifnum<1 \string#1}. The space at the end is important, because it will terminate the second number of the \texttt{\ifnum} if \texttt{#1} is a digit. The condition (\texttt{1 < 1}) is false if \texttt{#1} is a non-digit, and true (\texttt{1 < 10}, \texttt{1 < 11} etc.) otherwise. \texttt{\string} cancels the special catcode \texttt{1} might have. \texttt{#1} shouldn’t be longer than a single token, because the test makes \TeX{} process the extra tokens when \texttt{#1} contains a digit followed by extra tokens. If \texttt{#1} is \texttt{\if} or similar, the test isn’t skippable. The test works even if \texttt{#1} is empty.

The macro \texttt{\az} is also defined to insert a capitalized definite article at the beginning of a sentence. The macros \texttt{\aref, \Aref, \apageref, \Apageref, \cite} and \texttt{\Acite} are combinations of \texttt{\az} and referencing commands, so for example \texttt{\aref{foo}} is equivalent to \texttt{\az{\ref{foo}}}.

Counting Digits with Multiple Sentinels

A “sentinel” is something placed at the end of a list so that a conditional iteration over the list stops at the sentinel. For example, section 2.12 uses \texttt{\hfuzz} as a sentinel for the option processing of the macro \texttt{\processOptionsB}. A sentinel is usually a single token, but sometimes multiple sentinels have to be used in a row, when a macro processing them takes multiple parameters.

As mentioned in section 2.15, the Hungarian definite article (\texttt{a/az}) for a number depends on its pronunciation. The rule is: \texttt{az} has to be emitted for numbers starting with 5, and for numbers starting with \texttt{I} and having the number of digits following \texttt{I} divisible by 3. All other numbers are preceded by \texttt{a}. \texttt{magyar.ldf} thus contains a macro that counts number of digits following it:

\begin{verbatim}
\def\digitthree#1{\ifnum<1 \string#1 \ifnum<1 \string#2 \ifnum<1 \string#3
\def\digitthree{\numexpr\int#1\numexpr\int#2\numexpr\int#3
\ifnum9<1\string#1 \ifnum9<1\string#2 \ifnum9<1\string#3 % \else b\fi\else b\fi\else z\fi\endcsname}
\def\digitthree{b\if\else b\fi\else z\fi\endcsname}
\message{1:digitthree{} 100:digitthree{23+} 1000:digitthree{456}}
\end{verbatim}

In this example the macro \texttt{\digitthree} expands to \texttt{z} if its argument starts with digits of the multiple of 3. \texttt{\hbox$} is used as a sentinel to skip everything after the last digit has been found. The sentinel must not be present in the parameter itself. \texttt{\hbox$} makes no sense in \TeX{}, so it is quite reasonable to assume that the parameter doesn’t contain this. \texttt{magyar.ldf} uses \texttt{\hfuzz} and \texttt{\vfuzz} when only
a single token is allowed, because these two expan-
dable tokens are quite rare. The three consec-
utive slashes in the example are three sentinels, so
\texttt{digitthree@} has always enough arguments.

However, the test doesn’t work if the parameter of \texttt{digitthree} contains braces. For example
\texttt{digitthree\{1\{2x\}} would look for the undefined
control sequence \texttt{digitthree\{x\}}.

In the example the \texttt{csname} trick was used to
avoid \texttt{expandat} in the nested ifs. See the def-
ination of \texttt{@@namemagyar@az@set} in \texttt{magyar.ldf} for using
three multi-character sentinels in the same macro.

2.17 Definite Articles Before Roman
Numerals
\texttt{az} in \texttt{magyar.ldf} works differently for \texttt{az\{x\}} and
\texttt{az\{romannumeral 10\}} (see section 2.15), but
how should it distinguish when \texttt{romannumeral} has
already been expanded by the time \texttt{az} is called?
Although there is no general solution to the problem,
\texttt{magyar.ldf} addresses the case when \texttt{az\{ref\{my-
part\}\}} is called, having the label \texttt{my-part} point to a
\texttt{part}, when \texttt{def@thepart\{\@Roman@c@part\}} is
active. (This is so with the standard \texttt{book.cls}.) \texttt{ref}
gets the part number from the \texttt{\newlabel\{my-part\}
\{x\}\{42\}} command written to the .aux in the
previous run of \TeX. \texttt{label}, which has emitted this
\texttt{\newlabel} has already expanded \texttt{romannumeral}
in the previous run, long before our \texttt{ref} is called.

To make the definite article work in this spec-
cial case, \texttt{magyar.ldf} redefines \texttt{label} so it writes
\texttt{\hunnewlabel} in addition to \texttt{\newlabel} to the .aux
file. The arguments of \texttt{\hunnewlabel} are pre-expanded
when \texttt{\let\romannumeral\number} is in effect.
This solution also works when \texttt{\pageref} refers to a
roman numeral page number.

Expanding the page number at the right time
is rather tricky. The \TeX protected\@write says
\texttt{\let\thepage\relax}, which prevents expansion in
the following \texttt{edef}, so \texttt{\thepage} is expanded only
when the page is shipped out, and \texttt{\@page} contains
the right page number. What we want is to half-
expand \texttt{\thepage}, so it gets expanded to \texttt{\@Roman
\@page}, and \texttt{\@roman} is expanded to \texttt{\number} (!),
but the expansion of \texttt{\number\c@page} is postponed
until the page is shipped out. This can be done by
defining \texttt{\def\romannumeral\{\noexpand\number\}}
before calling \texttt{protected\@write}. In practice, \texttt{ma-
gyar.ldf} itself expands the page number, so three
\texttt{noexpands} are needed in front of \texttt{\number}.

Redefining \texttt{\label} (so it emits \texttt{\hunnewlabel})
also raises a problem. Some packages loaded later
might also override \texttt{\label}, for example \texttt{hyperref.sty}
loads \texttt{nameref.sty \AtBeginDocument}, which over-
rides \texttt{\label}. \texttt{magyar.ldf} recognises the new defi-
nition when the Hungarian language is activated —
which is done after the \texttt{\AtBeginDocument} hooks are
run (see section 2.21). So \texttt{\hunnewlabel} works fine
with \texttt{hyperref.sty}.

2.18 Removing All Braces
Removing all braces from a token list is required by
the \texttt{az} command (that inserts the \texttt{a/az} definite
article). \texttt{az} can find the first letter of its argument
even more easily if the argument doesn’t contain
braces.

The \texttt{removebraces} macro defined in figure 2
removes all braces and spaces (recursively) from
the tokens following it, until the first \texttt{h fuzz}. The
tokens may not contain a \texttt{h fuzz} inside braces, but
they may contain expandable material, even with
unbalanced conditional, because those are left un-
expanded in \texttt{removebraces@nobone} by \texttt{\noexpand}.
The most important trick here is the construct
\texttt{\ifcat\{\noexpand\#1\}} which is true if \texttt{\#1}
starts with a brace, and yields \texttt{\#1} with its first brace stripped.
\texttt{\\iffalse\fi} is needed so that the macro defi-
nition is nested with respect to braces. The usage of
\texttt{\@firstoftwo} is also worth mentioning: it is used to
change the \texttt{\removebraces@nobone} token following the
\texttt{\if} to \texttt{\removebraces}.

2.19 Changing \texttt{\catcodes Safely}
\texttt{\makeatletter} is equivalent to \texttt{\catcode 64 11}
on ASCII systems; this changes the category code
of characters having code 64 to 11 (letter). It is
possible to specify the character \texttt{0} without
knowing its character code: \texttt{\catcode'012}. Wherever
\texttt{\TeX} looks for a number (after \texttt{\catcode}, \texttt{\ifnum},
\texttt{\number}, etc.), it accepts a decimal number, an
octal number prefixed by \texttt{0}, a hexadecimal number
with digits \texttt{0-9A-F} prefixed by \texttt{`}, an internal counter
(such as \texttt{\linepenalty}), a \texttt{count} register (such as
\texttt{\count42} or \texttt{\oldlistdepth}) or a character prefixed
by \texttt{`}. The character can be specified as a character
token, or as a single-character control sequence.
It is wise to specify \texttt{\{ }, \texttt{\} } and space as \texttt{\{ }, \texttt{\}}, \texttt{\%} and
\texttt{\_\_}, respectively, so the whole construct is properly
nested with respect to braces, and since \texttt{\%} and space
tokens would be ignored.

However, many \texttt{Babel} language modules (.ldf
files) make the character \texttt{‘} active (i.e. \texttt{\catcode
13}), so the definition of \texttt{‘} in \texttt{\catcode@12} gets
expanded. The expansion can be prevented by
using \texttt{\noexpand}, but \texttt{\noexpand} yields \texttt{‘13}, which is
wrong, because \texttt{‘12} is needed, and moreover, will be
expanded in the second run, because \texttt{\TeX} is look-
ing for a number. Fortunately, \texttt{\string} solves the
problem, because \texttt{\string} changes the \texttt{\catcode} of

\newlabel{my-part}
the following character token to 12 (other) or 10 (space); and, if a control sequence follows, \string converts it to a series of character tokens with \catcode other or space.

Thus, the ideal definition of \makeatletter is \catcode\string`\@11\relax, which doesn't rely on the previous \catcode of ` or of @. The space at the end of the definition is needed so \TeX{} knows that the number 11 won't be followed by subsequent digits. Of course, the definition works only when the characters \catcodestring have \catcode letter, \ is an escape character (\catcode 0), and space is a space (\catcode 10). These are reasonable assumptions, because none of the standard \LaTeX{} packages change them.

The \LaTeX{} kernel's definition of \makeatletter is \catcode`\@11\relax, having \relax instead of space, which is equally good to mark the end of a number. This definition doesn’t need \string, because at the time it is read, the \catcode of ` is guaranteed to be 12 (other).

magyar.ldf saves the \catcode of ` ! & + - = | ; : , " / in the beginning, changes them to other, and restores them just before \endinput. This is needed in case other .ldfs have been loaded (e.g. \usepackage[french,magyar]{babel}) that have redeﬁned \catcodes. For example, french.ldf activates ! ? ; : .

It is also good not to change \catcodes until \begin{document} (not even \AtBeginDocument), because other packages not yet loaded may depend on the old, unchanged \catcodes. Babel, unfortunately, activates a character immediately when a shorthand is deﬁned in an .ldf ﬁle, so this can raise strange compatibility issues—which can be partly resolved by loading most other packages before Babel. magyar.ldf solves this by not touching the \catcode of its own shorthand at the time of deﬁnition, but instead calls \bb@activate in extrasmagyar, and \bb@deactivate in \noextrasmagyar. This is a local and temporary solution only. Future versions of Babel are expected to postpone character activation as far as \@preamblecmds (see also section 2.21).

2.20 Shorthands

A shorthand is an active character deﬁned by an .ldf ﬁle with the \declare@shorthand command provided by Babel. In this sense, all active punctuation characters (see section 2.9) are shorthands.

The most important shorthand in magyar.ldf is *13. (Most .ldf ﬁles choose that character to be the main shorthand, but some, such as germanb.ldf, choose *13.) The use of Hungarian shorthands can be disabled by the active= load option, and the shorthand character can be changed from * with the activeprefix= load option. magyar.ldf also provides the \shu* command, which is a longer form of *13, but without the possibly hazardous \catcode change.

Each shorthand is an active character, which raises compatibility problems (see section 2.19). magyar.ldf tries as hard as possible to avoid problems, but all efforts are in vain if another .ldf ﬁle is loaded which activates the same shorthand in the default (and unsafe) way.

For the user a shorthand is a control sequence without a backlash, so a shorthand is a command that can be typed and read quickly. germanb.sty provides "a to be equivalent to \"a, saving a keystroke for every accented German letter. magyar.sty doesn't provide this saving, because the letters o and u have 3 accented forms, and introducing different letters for them would lead to confusion. Hungarian \LaTeX{} authors are encouraged to use the latin2 encoding to type accented letters as a single character.

But the shorthand does an important job concerning (unaccented) long double consonants; for example, 'tty is an abbreviation for \v{t}no\break\discretionary{y-}{ }{ }ty\nobreak\hskip\vskip. (Section 2.6 explains why this is needed.) It should
be noted that shorthands are implemented as \TeX macros, so ‘\texttt{tty} and ‘\texttt{tty} are equivalent.

The shorthand functionality of magyar.ldf for non-letters is inspired by ukraineb.ldf. ‘* and ‘- stand for a hyphen that separates words, so both words are automatically hyphenated by \TeX (implemented as \texttt{\leavevmode\nobreak\hskip\z@skip}; ‘* in math mode stands for a space character following a delimiter (\texttt{\mskip2.4mu plus3.6mu minus1.8mu}) that will magically be exactly as wide as if a space was inserted outside math mode, because the implicit \mskip0.6mu after the delimiter is already subtracted; ‘-- emits \texttt{,\textendash{}}, to be used between author names in Hungarian bibliographies; ‘\ allows a hyphen that is repeated at the beginning of the next line if the line is broken there (implementation: \leavevmode\nobreak\hskip\z@skip\nobreak\hskip\z@skip); to be used with long words (e.g. nátrium–klorid) having important hyphens: ‘_ inserts a discretionary hyphen with automatic hyphenation enabled at both sides; ‘< inserts a French opening guillemet even if the ligature ‘< is missing from the current font; ‘> inserts its paired closing; ‘’ is equivalent to \texttt{\allowbreak} with hyphenation enabled on both sides (implementation: \texttt{\mskip\z@skip\nobreak\hskip\z@skip}); ‘‘ inserts a hyphen that doesn’t form ligatures when repeated (implementation: \texttt{\leavevmode\hbox{\{-\}}}{}.

2.22 Displaying Theorem Titles

In English, theorem titles are displayed as “Theorem 1”, but Hungarian requires “1. tétel.”. To implement this, the \texttt{\begin{theorem}} and \texttt{\opargbegin theorem} macros are redefined each time the Hungarian language is activated. However, if \texttt{theorem.sty} or \texttt{ntheorem.sty} is loaded, the changes have to be embedded into a theorem style. The chosen name for the style is \texttt{magyar-plain}. It is activated by default when \texttt{magyar.ldf} is loaded, so theorem titles will come out right unless the user calls \texttt{\select@language{magyar-plain}}. When \texttt{amsthm.sty} is loaded, \texttt{magyar.ldf} redefines the macros \texttt{\paragraph} and \texttt{\swappedhead} so both will emit the title properly.

2.24 Indentation after Section Titles

Hungarian typography allows the first paragraph after a section title to be either indented or unindented, so \texttt{magyar.ldf} provides \texttt{afterindent=} as the load option to control this. \TeX calculates the value of a boolean variable \texttt{\if@afterindent} from
The Decimal Comma

The dot character is defined as ordinary in mathematical text by default, so decimal real numbers can be typed simply as \$-12.34\$. Hungarian denotes the decimal point by a comma instead of a dot, but typing \$-12,34\$ yields ‘$-12\,34’ with too much space after the comma, because the comma is defined as punctuation rather ordinary in math text. \$-12{,}34\$ yields ‘$-12\,34’, which is correct, but \texttt{magyar.ldf} provides two mechanisms to save the two keystrokes of the curly braces around the comma.

First, the \texttt{HuComma} macro below inserts an ordinary comma if it is followed by a digit, and an operator comma otherwise:

\begin{verbatim}
\edef\hucomma@lowa#1#2 #3#4 #5#6\hfuzz{\%}
  \noexpand\if\ifnum9<1#5 \noexpand\if\if#1t\noexpand\if\if#3c\noexpand\if\if\if\if\if#6\noexpand\fi\else\noexpand\fi\else\noexpand\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
\end{verbatim}

\begin{verbatim}
\HuComma
{\futurelet\reserved@a\hucomma@lowb}
\end{verbatim}

The solution Donald Arseneau proposed to the comma problem inspired these macros. In line (1) \texttt{hucomma@lowa} tests whether the \texttt{meaning} of the following character is ‘the character (digit)’. A \texttt{meaning} is always at least three words, but it may be more (\texttt{e.g. ‘math shift character ’}). Only the \texttt{character} starts with letters \texttt{t} and \texttt{c}. An \texttt{edef} is needed above so the \texttt{mathchar} emitted doesn’t depend on the \texttt{mathcode} changes after the definition of \texttt{HuComma}. Then the comma character can be redefined as \texttt{HuComma}, as given in figure 3.

With these definitions, the formula ‘\(F_i(x,y) = y^i + 1.3x, \ x, y \in A, \ i = 1, 2, 3, \ldots\)’ can be typed simply as \$F_i(x,y) = y^i + 1.3x, \ x, y \in A, \ i=1, 2, 3, \ldots\$. If $\mathring{\iota}$ is breakable (such as in \texttt{nath.sty}).

When \texttt{nath.sty} is loaded, the definitions are appended to \texttt{mathoptions@on}, and if \texttt{nath.sty} is missing, to \texttt{check@mathfonts}. The appropriate macro is run just before \texttt{everymath} by \LaTeX{}. Redefining the \texttt{mathcode} and \texttt{mathcode} this way ensures that the proper comma is used inside math mode — unless the whole math formula is a macro argument with already assigned \texttt{mathcode}s. Also, it is not a good use of \texttt{begingroup}, \texttt{iccode}, \texttt{lowercase} and \texttt{endgroup} to modify the active meaning of a character without actually activating it. Calling \texttt{mathcode}, 13 before \texttt{def} wouldn’t help here anyway if the whole construct is embedded into a macro definition, because \texttt{mathcode} wouldn’t be able to change an already assigned catcode.

\texttt{frenchb.ldf} provides \texttt{DecimalMathComma} and \texttt{StandardMathComma} to change the \texttt{mathcode} of the comma. However, the smart comma based on \texttt{HuComma} acts correctly without the user needing to be aware of curly braces or redefinitions.

The solution above can be activated with the loading option \texttt{mathcomma=fix}. An alternative approach doesn’t alter \texttt{mathcodes}, but introduces a special math mode in which the dot appears as a comma only when the Hungarian language is active. Thus the printout of \texttt{MathReal{−12.34}} depends on the current \texttt{Babel} language. The definition of \texttt{MathReal} in \texttt{magyar.ldf} is similar to:

\begin{verbatim}
\def\mathreal@lowb#1.{%\mathchar"013B \\mathreal@lowb}}% comma
\def\mathreal@lowa#1{\ensuremath{%\\mathchar"013B \\mathreal@lowb}}% comma
\end{verbatim}

The argument of \texttt{MathReal} must contain the dot to be changed literally, outside braces. There is a little macro wizardry in the implementation that stops calling \texttt{mathreal@lowb} infinitely. The call \texttt{mathreal@lowa} terminates its argument by a sentinel \texttt{@gobble}, so \texttt{#1} of \texttt{mathreal@lowb} will end by \texttt{@gobble}, which will gobble \texttt{@secondoftwo}, so the \texttt{@gobble} in line (1) will take effect, which stops the recursion.

\texttt{MathReal} is going to be extended in the future so it will handle physical units following the number properly, and it will also insert thin spaces after each three digits. This feature has already been implemented in \texttt{frenchb.ldf}.
2.26 Parsing Dates

There are many correct ways to write dates in Hungarian, and magyar.ldf provides an \emitdate command that can generate any of these formats. Doing the reverse is a little more interesting.

Let’s suppose we have a Gregorian date consisting of a year (4 or 2 digits), a month (a number or a name) and a day-of-month in some standard format. We want a command \parsedate to detect the format, split the date into fields, and call \fixdate:

\def\fixdate#1#2#3{\% 1. \let\today{\the\year-\the\month-\the\day}%ISO 2. \let\protect\string \% remove accents from Hungarian month names: 3. \let\@firstofone \@secondofone 4. \let~\space \%change '2003.~okt' to '2003. okt' 5. \edef\re@b{\def\noexpand\re@b{#1}}\% \expandafter\endgroup\re@b \edef\re@b{\expandafter\stripdot\expandafter\re@b}\% \let\re@a\@empty \expandafter\parsedate@a\re@b \ifx\re@a\@empty \expandafter\parsedate@f\re@b !//:!\hfuzz \fi \ifx\re@a\@empty \expandafter\parsedate@b\re@b !//!\hfuzz \fi \ifx\re@a\@empty \expandafter\parsedate@c\re@b !..!\hfuzz \fi \ifx\re@a\@empty \expandafter\parsedate@d\re@b !. xyz !\hfuzz \fi \ifx\re@a\@empty \expandafter\parsedate@e\re@b !xyz , !\hfuzz \fi \ifx\re@a\@empty \errmessage{Unrecognised date: \re@b}\% \else \re@a% call \fixdate \fi}

Figure 4: \parsedate: Parse date formats.

Many dates have an optional dot at the end. Since that dot doesn’t carry useful information, we should remove it first. The \stripdot command defined below expands to its argument with the trailing dot removed. \stripdot works only if the argument doesn’t contain the token \relax. \relax is not special; any other token would have worked.

\def\stripdot#1{\expandafter\stripdot@lowb\stripdot@lowa#1\relax.\relax}\
\def\stripdot@lowa#1.\relax{#1\relax}\
\def\stripdot@lowb#1\relax#2\relax{#1}\%

The definition of \parsedate is shown in figure 4. \parsedate first does some generic cleanup, and puts the resulting date into \re@b. \endgroup cancels the redefinition of \today etc., but \re@b is expanded first, which defines itself, so the value of \re@b will be retained after \endgroup. After that, the trailing dot is stripped, and then various \parsedate@... commands are run. If a command recognises the date format, it puts a call to \fixdate into \re@a, which will be called at the end of \parsedate. Strange strings like !:///!\hfuzz are sentinels.

The idiom \expandafter\endgroup\re@b is an important trick for expanding a macro before the current group completes (and changes are undone). It usually contains definitions of other control sequences whose meanings are about to be retained after the end of the group. An alternative would be to inject such a definition using \aftergroup, but that only accepts a single token, so it would be very painful to make a macro definition with spaces and braces survive this way.

The individual \parsedate@... commands are given in figure 5. This implementation of date parsing isn’t error-proof. If something weird is passed to \parsedate, it may produce surprising \TeX errors. However, \parsedate can distinguish between different formats of correct input.

2.27 Setting Up French Spacing

Hungarian typography requires \frenchspacing to be turned on, but most \TeX users fail to follow this requirement. Babel provides the command \bbl@frenchspacing, which turns French spacing on if it was off. The frenchspacing= load option of magyar.ldf controls how Hungarian text should behave. For the sake of symmetry, magyar.ldf provides \@@magyar@antifrenchspacing, which—contrary to the typographical requirement—turns french spacing off:

\def\@@magyar@antifrenchspacing{% 1. \ifnum\the\sfcode'.=\@m \nonfrenchspacing \let\@@magyar@nonfrenchspacing\frenchspacing \else \let\@@magyar@nonfrenchspacing\relax \fi} \addto\extrasmagyar{\@@magyar@antifrenchspacing}% Preprints for the 2004 Annual Meeting

\def\parsedate@1{% \begingroup \def\today{\the\year-\the\month-\the\day}\%ISO \let\protect\string \% remove accents from Hungarian month names: \let\@firstofone \@secondofone \let~\space \%change '2003.~okt' to '2003. okt' \edef\re@b{\def\noexpand\re@b{#1}}\% \expandafter\endgroup\re@b \edef\re@b{\expandafter\stripdot\expandafter\re@b}\% \let\re@a\@empty \expandafter\parsedate@a\re@b \ifx\re@a\@empty \expandafter\parsedate@f\re@b !//:!\hfuzz \fi \ifx\re@a\@empty \expandafter\parsedate@b\re@b !//!\hfuzz \fi \ifx\re@a\@empty \expandafter\parsedate@c\re@b !..!\hfuzz \fi \ifx\re@a\@empty \expandafter\parsedate@d\re@b !. xyz !\hfuzz \fi \ifx\re@a\@empty \expandafter\parsedate@e\re@b !xyz , !\hfuzz \fi \ifx\re@a\@empty \errmessage{Unrecognised date: \re@b}\% \else \re@a% call \fixdate \fi}
\def\parsedate@a{\texttt{#1-#2-#3!#4}}\hfuzz{\% ISO date: YYYY-MM-DD}
\ifx\hfuzz\texttt{#4}\hfuzz\else 
\ifnum1<1\string#1\relax 
\ifnum1<1\string#2\relax 
\ifnum1<1\string#3\relax 
\def\re@a{\texttt{fixdate{#1}{#2}{#3}}}\% 
\fi\fi\fi\fi}

\def\parsedate@b{\texttt{#1/#2/#3!#4}}\hfuzz{\% LaTeX date: YYYY/MM/DD}
\ifx\hfuzz\texttt{#4}\hfuzz\else 
\ifnum1<1\string#1\relax 
\ifnum1<1\string#2\relax 
\ifnum1<1\string#3\relax 
\def\re@a{\texttt{fixdate{#1}{#2}{#3}}}\% 
\fi\fi\fi\fi}

\def\parsedate@c{\texttt{#1.#2.#3!#4}}\hfuzz{\% English date: YYYY.DD.MM}
\ifx\hfuzz\texttt{#4}\hfuzz\else 
\ifnum1<1\string#1\relax 
\ifnum1<1\string#6\relax 
\lowercase{\%}
\expandafter\ifx\csname mon@#2#3\endcsname\relax
defun{\re@a{\texttt{fixdate{\number#1}{\csname mon@#2#3\endcsname}{\number#6}}}}\fi)
\fi\fi\fi\fi)

\def\parsedate@d{\texttt{#1. #2#3#4#5 #6!#7}}\hfuzz{\% \{2003. oktober 25\}}
\ifx\hfuzz\texttt{#7}\hfuzz\else 
\ifnum1<1\string#1\relax 
\ifnum1<1\string#6\relax 
\lowercase{\%}
\edef\re@a{\texttt{fixdate{\number#6}{\csname mon@#1#2#3\endcsname}{\number#5}}}\fi\fi\fi\fi)

\def\parsedate@e{\texttt{#1#2#3#4 #5 #6!#7}}\hfuzz{\% \{October 25, 2003\}}
\ifx\hfuzz\texttt{#7}\hfuzz\else 
\ifnum1<1\string#5\relax 
\ifnum1<1\string#6\relax 
\lowercase{\%}
\edef\re@a{\texttt{fixdate{\number#6}{\csname mon@#1#2#3\endcsname}{\number#5}}}\fi\fi\fi\fi)

\def\parsedate@f{\texttt{#1/#2/#3:#4!#5}}\hfuzz{\% LaTeX default \today}
\ifx\hfuzz\texttt{#5}\hfuzz\else 
\ifnum1<1\string#1\relax 
\ifnum1<1\string#2\relax 
\ifnum1<1\string#3\relax 
\def\re@a{\texttt{fixdate{#1}{#2}{#3}}}\% 
\fi\fi\fi\fi)

\begin{figure}[ht]
\centering
\includegraphics[width=\textwidth]{figure5.png}
\caption{Individual \texttt{parsedate} commands.}
\end{figure}
2.28 varioref.sty Fixes

The magyar load option of varioref.sty (2001/09/04 v1.3c) is buggy, because it uses the never-defined \aza command for adding definite articles, and it also calls \AtBeginDocument too late, producing a \hbox error each time the Hungarian language is activated. magyar.ldf contains the correct definitions for the language-specific text reference macros, such as \ref\textlabelrange, and also contains ugly fix-up code to remove the wrong macros inserted by varioref.sty. A patch has been sent recently to the author of varioref.sty.

Some of these text reference macros use the \az and the \@old commands defined by magyar.ldf.

2.29 Removing Full Stops After Section Titles in AMS Classes

AMS document classes always append a full stop after section titles, which is strictly forbidden in Hungarian typography. The solution is to remove the tokens \addpunct. from the definition of \@sect (and also from \NR@sect in case nameref.sty has also been loaded). But this simple idea is quite complicated to program, and the result is ugly, as seen in figure 6. This detects AMS classes by the presence of \global\nobreaktrue\@xsect in the definition of \@sect, and adds code just before \@xsect. The code added prepends \let\addpunct\@gobble to the definition of \@svsechd. \@svsechd is later called by \@xsect, which calls \addpunct, but by that time \addpunct is a no-op. The application of this fix is controlled by the amspostsectiondot= load option.

2.30 Reduced Math Skips

Investigations in [1] have shown that the following settings produce the desired space for Hungarian math mode:

\thickmuskip 4mu plus 2mu minus4mu
% LaTeX: ditto
\medmuskip 2mu plus1.5mu minus2mu
% LaTeX: 4mu plus2mu minus4mu
\thickmuskip 3mu % LaTeX: 5mu plus5mu minus5mu

Notice that \medmuskip < \thickmuskip. These settings can be selected in magyar.ldf with the load option mathmuskips=. The difference between the original and the reduced spacing:

\[
a + b - c/d \cdot y \circ x = z \quad a + b - c/d \star y \circ x = z
\]

2.31 Breaking a Long Inline Math Formula

Hungarian typography requires that a binary relation or operator (e.g. in 1 + 2 = 3 + 4) must be repeated in the next line if an inline math formula is broken there. This can be accomplished for the equation sign by substituting \nobreak\discretionary{}{\hbox{(\!*\)}}{} for \* in math formulas. The long inline formula delimiters \( and \) \) are used because the catcode of the \$ would be wrong if nath.sty was loaded after magyar.ldf. \nobreak is necessary, so \TeX itself won’t break the line after the =.

The mathbrk= load option of magyar.ldf controls whether the operators and relations should be redefined. If so, the operators +, −, ∗ (as well as 37 operators available as control sequences) and the relations <, >, =, : (as well as 43 relations available as control sequences) are modified so they get repeated at the beginning of the line. The \\dot and the \slash operators are also modified, because Hungarian typography disallows breaking the line around them.

2.32 Restarting Footnote Numbering on Each Page

Although \usepackage[perpage]{footmisc} and footnpag.sty provide these features, magyar.ldf allows normal arabic footnote and page-restarting asterisk-foottnotes to be intermixed. It is common in Hungarian article collections to have the notes of the author numbered in arabic (by \footnote), and the footnotes of the editor added with asterisks (by \editorfootnote). The first four editorial footnotes on a page are marked with *, **, ***, and †. magyar.ldf also inserts proper additional space between the footnote mark and the footnote text, and the footnote facility is fully customizable with the \footnotestyle command.

The basic idea behind the implementation of pagewise numberering is creating a \label for each footnote, and whenever the \pageref for that label shows a different page, resetting the counter to zero. This clobbering can be automated by abusing the \cl@footnote hook. Each time \stepcounter advances a counter, the corresponding \cl@... hook is called, which usually resets other counters (for example, advancing the chapter counter resets the section counter). But arbitrary code can be executed after the automatic \stepcounter\{footnote\} by appending that code to the macro \cl@footnote.

The famous problem of creating a macro that will expand to n asterisks is proposed in appendix D of The \TeXbook [3]. David Kastrup has provided
Figure 6: Removing full stops after AMS section titles.

a brilliant solution to the problem in \cite{2}, namely \
\texttt{\expandafter\mtostar\romannumeral
\number000A}, where \texttt{\mtostar} transforms ms to asterisks: \texttt{\edef\mtostar#1{\if#1m*\expandafter\mtostar\fi}}. This solution is used in \texttt{magyar.ldf}.

\texttt{magyar.ldf} also provides the following command to insert footnotes into section titles such that neither the table of contents nor the page headings are affected:

\texttt{\def\headingfootnote{\ifx\protect\@typeset@protect\expandafter\footnote\else\expandafter\@gobble\fi}}

2.33 Class-specific Modifications

\texttt{magyar.ldf} does some modifications based on the current document class (using the \texttt{\@ifclassloaded} \LaTeX\ command). Only the standard classes \texttt{article.cls}, \texttt{report.cls}, \texttt{book.cls} and \texttt{letter.cls} are supported at present. The visual appearance of the \texttt{\part} and \texttt{\chapter} output is changed, and the page headers are also modified. For \texttt{book.cls}, part numbering is spelled out, so “Part 1” becomes “Első rész” (“Part One”) if the load option \texttt{partnumber=} is set to \texttt{\Hunordinal}.

The command \texttt{\ps@headings} has to be executed again to install its changed heading macros. This is called from \texttt{\@preamblecmds}, after the default language has been activated (see section 2.21).

The typographically correct customization of \texttt{letter.cls} is under development.

2.34 Spelling Out Numerals and Ordinals

The \texttt{\@Hunumeral} and \texttt{\@Huordinal} macros defined in \texttt{magyar.ldf} can spell out integers between $-9999$ and $9999$. \texttt{\@Hunumeral} and \texttt{\@Huordinal} are the capitalized versions of these macros. For example, \texttt{\@huordinal{2004}} produces \texttt{kétezer-négyedik} (“two thousand and fourth”) and \texttt{\@Hunumeral{2004}} produces \texttt{Két-ezer-négy} (“Two thousand and four”). All of these macros are fully expandable, so they can be used for \texttt{\part} numbering: \texttt{\def\thepart{\@huordinal\c@part}}, or more simply: \texttt{\def\thepart{\Huordinal{part}}}.

The most important implementation issue is the method to retrieve the last digit of a number in an expandable construct. If the number is between 0 and 9999, the following macro solves the problem:

\texttt{\def\LastDigitOf#1{\expandafter\lastdigit@a
\number#1;}}
\texttt{\def\lastdigit@a#1;{% #1 in 0..9999
\ifnum#1<10 #1\else\ifnum#1<100 \lastdigit@b00#1%\else\ifnum#1<1000 \lastdigit@b0#1%\else\lastdigit@b#1\fi\def\lastdigit@b#1#2#3#4{#4}\fi}}

2.35 Suffix Generation

As mentioned earlier, the Hungarian language has suffixes to represent relations in space and time, instead of prepositions. For example, an English math text might contain “It follows from (1),” in which “from (1)” can be typed as \texttt{from (\ref{eq1})}. The \LaTeX\ referencing scheme guarantees that the text above will come out right, even if the order of equations is changed in the document.

But in Hungarian, the suffix standing in place of “from” has two forms: -ból/-ből, depending on the vowel harmony of the pronunciation of \texttt{\ref{eq1}}. So there is a need for automatic suffix generation.
magyar.ldf provides the command \told, with which the Hungarian version of “from (1)” can be typed as \told((\ref{eq1})+bol{}), which will generate “(1)-ből”, “(2)-ből”, but “(3)-ből”.
\told can handle 20 different suffixes, and 4-20 suffix combinations (such as \told3+adix+ra{}), meaning “to the third”). Only the last number is considered in references containing multiple numbers. Roman numerals are recognised properly in references with the help of \hunnewlabel (see section 2.17) — this is implemented similar to \az. Suffix generation is supported only for integers and Hungarian document structure names (see section 3.2), because writing a generic suffix generator without a database is quite a difficult task, and definitely won’t give Hungarian LTEX users much more comfort than the current \told implementation.

Although most Hungarian suffixes have 1, 2 or 3 forms,\footnote{Of course, suffixes with only one form are not supported by \told.} numbers can be classified into 23 paradigm classes, so that the paradigm class uniquely determines the correct form of all known suffixes. The reason that there are so many classes is because the letter \textit{v} of the -\textit{val}/-\textit{vel} suffix must be also changed to the last letter of the number if that letter is a consonant. Essentially each final digit has a class, and there are classes for the powers of 10, and some of the numbers 20, 30, . . . 90 also have their own classes. To sum up, the suffix of a number depends on the last nonzero digit, and the number of trailing zeroes.

The implementation of \told is surprisingly long and ugly, full of recursive macros that parse the input, and it doesn’t contain any bright ideas that are not also found elsewhere in magyar.ldf. The curious LTEX hacker should study \az instead, because it is shorter and its trick density is much higher.

2.36 Warning Messages

magyar.ldf has the unique feature that it displays warning messages (called ‘suggestions’) at load time to notify the user that they are using magyar.ldf in a possibly incorrect way. If they are not disabled by the \texttt{suggestions=load} option, the following suggestions are displayed to standard output during the \texttt{AtBeginDocument} hook:

- the user forgot to load \texttt{\usepackage{tienc}} — so words with accented letters won’t hyphenate automatically;
- the user forgot to load \texttt{\usepackage[latin2] {inputenc}}, or the input encoding chosen is not latin2, cp1250 or utf8 — so there is a good chance that accented characters will disappear or come out wrong;
- the Hungarian hyphenation patterns requested were not found — magyar.ldf tries to use the other two possible Hungarian patterns, if they are available;
- \texttt{\def\magyarOptions or \PassOptionsToPackage{}{magyar.ldf}} was specified too late — late options can be detected, but they have no effect, since options do their work while magyar.ldf is being \texttt{loaded};
- the buggy \texttt{varioref.sty} has been loaded as \texttt{\use package[magyar]{varioref}} — this will happen until the patch is integrated to \texttt{varioref.sty}; the current version is so buggy that it displays an untraceable LTEX error each time the Hungarian language is activated (see section 2.28).

3 Miscellaneous Tricks

First we show some common expansion tool macros defined by LTEX:

\begin{verbatim}
\def\@empty{}
\long\def\@gobbletwo#1#2{}
\long\def\@firstoftwo#1#2{#1}
\long\def\@secondoftwo#1#2{#2}
\long\def\@gobble#1{}
\def\@empty{}
\end{verbatim}

\texttt{\@empty{}}, because it may not be followed by \texttt{)}, it ignores spaces in front of its argument, and it removes at most one pair of braces around its argument. All of these properties are consequences of the macro expansion rules described in chapter 20 of \textit{The \TeX{}book} [3].

This remainder of this section describes \TeX{} macro and typesetting tricks not tightly related to the Hungarian language.

3.1 The Factorial Sign in Math Mode

nath.sty contains a smart definition of the factorial operator, so \( (a+b)!/a!\cdot b! + c! \cdot d! \), with proper spacing can be typed as $$\frac{(a+b)!}{a!\cdot b!}+c!\cdot d!$$.

The only place where \texttt{ből} needs is before the slash. magyar.ldf adapts the definition:

\begin{verbatim}
\def\factorial{\mathchar"5021\mathopen{}%}
\end{verbatim}

\texttt{\@gobbletwo} differs from \texttt{\@empty}, because it may not be followed by \texttt{)}, it ignores spaces in front of its argument, and it removes at most one pair of braces around its argument. All of these properties are consequences of the macro expansion rules described in chapter 20 of \textit{The \TeX{}book} [3].

This remainder of this section describes \TeX{} macro and typesetting tricks not tightly related to the Hungarian language.
Figure 7: \nonumbers: Change all roman and arabic numbers to 1.

3.2 Including the Structure Name in References

Text like “in subsection 5.6” is usually typed as in subsection\textit{ref{that}}. But it would be nice if \LaTeX\ were able to guess that \texttt{ref{that}} actually points to a subsection. The subsection depth can be deduced by counting the dots in the reference: a subsection has one dot (in an article), and a subsubsection has two dots.

\texttt{magyar.ldf} provides the \texttt{\refstruc} command which has a smarter detection scheme: it changes all roman and arabic numbers to one (1, 1 or I) in the reference, and compares the result with the tokens generated by \texttt{\thechapter}, \texttt{\thesection} etc., with \texttt{\chapter} etc. set to 1 temporarily. This should work in most cases, although it cannot refer to equations, tables or figures. In Hungarian text, the Hungarian structure names are emitted, and other texts the original, English control sequence names are printed. \texttt{\refstruc} includes the definite article and suffixes support; for example, \texttt{\Az{\refstruc{that+tol}}} may emit az I. fejezet\texttt{túl} (“from chapter 1”).

The full implementation is quite long, and is not included here, but the macro that changes all roman and arabic numbers to one is presented in figure 7.

3.3 Enabling Long Page Numbers

If the width of a page number in the table of contents is greater than \texttt{@pnumwidth}, \LaTeX\ emits an “Overfull \texttt{\hbox}” warning. This can be eliminated by changing \texttt{@dottedtocline} in the \LaTeX\ kernel. The line \texttt{\hbox{\@pnumwidth}{\hfil \normalfont \normalcolor #5}} should be changed to:

\texttt{\hbox{\@pnumwidth}{\hfil \normalfont \normalcolor #5}}

Although this change isn’t related to the Hungarian language, \texttt{magyar.ldf} will do it given the \texttt{\dottedtocline=load} option.

3.4 Removing AMS Warnings from \texttt{\listoftables} and \texttt{\listoffigure}

Some AMS document classes (such as \texttt{amsart.cls}) produce an “Overfull \texttt{\hbox}” warning for each line in the \texttt{\listoftables} and \texttt{\listoffigure}. This can be fixed by changing this line in the \texttt{\@table} and \texttt{\@figure} macros in the AMS classes:

\texttt{\@tocline{0}{3pt plus2pt}{0pt}{\parindent}{}}

\texttt{\@tocline{0}{3pt plus2pt}{0pt}{\parindent}{}}

The code shown in figure 8 makes this change.

The control sequences \texttt{\allowttbyphens} and \texttt{\setTrue} are defined by each of the AMS document classes, so their presence indicates that one of those classes is loaded. The logical or operation using \texttt{\ifx} is also worth noting.

3.5 Discarding to End of File

The naïve solution \texttt{\ifskiprest\endinput\fi} results in the \LaTeX\ error message “end occurred when \texttt{\iftrue} in line $n$ was incomplete” if there is a line
break at the ¶ sign. Without the line break, the naïve solution works perfectly, because \endinput stops reading the current file after the current line, so the \fi also gets evaluated.

It is possible to do something before \endinput:
\begin{verbatim}
\begin{tikzpicture}
\node (a) at (0,0) {A};
\node (b) at (1,0) {B};
\end{tikzpicture}
\end{verbatim}

All of the above must be put after \endinput without a line break. The \texttt{csname fi} \texttt{endcsname}
construct closes the \texttt{if} when the condition is true, but is invisible when the condition is false, and \TeX's
is skipping tokens.

The \texttt{\LaTeX} kernel macro \texttt{@ifpackageloaded} implements the conditional end by a different trick. The following two constructs are equivalent:
\begin{verbatim}
@ifpackageloaded{foo}{\errmessage{I am incompatible with foo.sty}}{}
\end{verbatim}

and
\begin{verbatim}
\begin{tikzpicture}
\node (a) at (0,0) {A};
\node (b) at (1,0) {B};
\end{tikzpicture}
\end{verbatim}

In the trick above, \texttt{\errmessage} isn't on the same line as \texttt{\endinput}. This isn't a problem, because by the time \texttt{\endinput} is evaluated by \TeX's stomach, \texttt{\errmessage} has been read from the file, and it is already on the input stack. \texttt{\endinput} doesn't discard the input stack, it just prevents more lines from being read from the current file.

3.6 Typesetting Text Verbatim

The change is completely avoided using a different approach, based on macro arguments:
\begin{verbatim}
\if\noexpand#1b\helpif\fi\@secondoftwo {B}
\else \if\noexpand#1aA\helpif\fi\@secondoftwo {A}
\fi\message{ucabs cbbas} \% -> cBcB
\end{verbatim}

The construct doesn't work when \#1 has braces around it, or it is \texttt{\if\ldots,\else or \fi}. Also, spaces will be ignored because of macro expansion.

But what if we'd like to capitalize only the first \texttt{a} or \texttt{b}? Then we would need \texttt{\expandafter\expand after\expandafter\gobble} after \texttt{A}, and seven \texttt{\expandafter\expand after\expandafter\gobble} after \texttt{B}. But \texttt{\expandafter} can be completely avoided using a different approach, based on macro arguments:
\begin{verbatim}
\def\helpif#1\#2{\@firstoftwo{A \#1}}
\def\ucabs#1{}\message{ucabs cbbas} \% -> cBcB
\end{verbatim}

It is not possible to move \texttt{\if} into the definition of \texttt{\helpif}, because then \TeX won't see that particular \texttt{\fi} when it is skipping the whole \texttt{\if} \ldots \texttt{\helpif} construction. With a small rearrangement we can get rid of \texttt{\@secondoftwo}:
3.9 Processing Arbitrary Package Options

\%TEX packages can use the standard commands \texttt{\textbackslash DeclareOption, \textbackslash ExecuteOption and \textbackslash ProcessOptions} to access package options passed to them, and these commands work fine with a fixed set of options. The \texttt{\textbackslash DeclareOption*} command can be used to declare arbitrary options:

\%\textbackslash DeclareOption{10pt}{\texttt{\textbackslash typeout(got ten-pt)}}%\textbackslash DeclareOption*{\texttt{\textbackslash typeout(got=\textbackslash CurrentOption)}}\% in file foo.sty

Two lines will be printed when foo.sty is loaded as \texttt{\textbackslash usepackage[\texttt{foo=bar, no,}]\{foo\}}. These are got=(\texttt{foo=bar}) and got=no. The optional argument of \texttt{\textbackslash usepackage} may contain spaces and/or a single newline around commas and at the ends. Class options are passed to \texttt{\textbackslash DeclareOption*}, so when \texttt{\textbackslash documentclass[10pt]{article}} is active, got=(10pt) will not appear, but when line (1) is uncommented, got ten-pt will appear.

There is an alternative, low-level way for accessing all the options at once:

\%\texttt{AtEndOfPackage{\textbackslash let@unprocessedoptions \relax}\% prevent warning \textbackslash typeout{\csname opt@\@currname.\@currext \endcsname}}

This prints the full option list with extra spaces and newlines removed, but commas, including superfluous ones, are kept intact.

4 Beyond the Current magyar.\ldf

4.1 Other Hungarian Typesetting Software

Although magyar.\ldf contains most of the functionality needed for following Hungarian typographic traditions, other utilities and packages can help in typesetting Hungarian texts. Most of this software, including magyar.\ldf, is going to be available under the name Magyar.\TeX from \url{http://www.math.bme.hu/latex/}.

magyar.\ldf The new Hungarian module for Babel. Version 1.5 was written by Péter Szabó beginning in the autumn of 2003.

huhyph.\tex or huhyph3.\tex The old (version 3) Hungarian hyphenation \texttt{patterns} for the T1 encoding, written by Gyula Mayer in 1998. Part of most \TeX distributions. See section 2.1.

huhyphc.\tex The new version of the Hungarian hyphenation \texttt{patterns} for the T1 encoding, written by Gyula Mayer in 2002 [4]. Part of most \TeX distributions. Hyphenates foreign compound words on the subword boundary, e.g. szin-kron. See section 2.1.
The new version of the Hungarian hyphenation \texttt{patterns} for the T1 encoding, written by Gyula Mayer in 2002 \cite{Mayer2002}. Part of most \TeX\ distributions. Hyphenates foreign compound words phonetically, e.g. \texttt{szink-ron}. See section 2.1.

\texttt{ccs\_extract.pl} A Perl script that helps with hyphenation of words containing long double Hungarian consonants. It finds all occurrences of such words in the document and lets the user decide whether to insert, for each unique word, the Hungarian shorthands for long double consonants. The program was written by Péter Szabó in 2003.

\texttt{lafmgen.pl} An easy-to-use Perl script that can generate format files (.fnt) containing all hyphenation patterns required by the specified \TeX\ document. It has some other features related to generating and installing format files, and is to be used with the Unix te\TeX\ distribution. It was written by Péter Szabó in 2003.

\texttt{huplain.bst} Bin\TeX\ style file for Hungarian bibliographies, based on plain.bst. It encourages sharing the same .bib database between Hungarian and English documents. It follows the (simple) convention that the style of a bibliography depends on the language of the document containing the entry, not the language of the entry itself. It was written by Péter Szabó in 2003.

\texttt{husort.pl} A drop-in replacement of makeindex that follows the Hungarian standards of index sorting and typesetting. It is implemented as a Perl script. It was written by Péter Szabó in 2003.

\texttt{magyar.xdy} Hungarian style file for the \Xindy\ index processing program. Implements the Hungarian sorting order and a Hungarian typesetting style. The implemented sorting order does not follow Hungarian rules as strictly and elegantly as husort.pl. It was written by Péter Szabó in 2003.

\texttt{CM-Super} The EC fonts in Type 1 format in T1 and various other encodings. It is not part of Magyar\TeX, but is available from CTAN. It is useful for converting Hungarian text to PDF, so the generated PDF file will contain the EC fonts in Type 1 format, and will be rendered quickly and nicely by Acrobat Reader.

MagyarSpell The Hungarian language database of the Ispell spell checker for Unix. On Debian systems, it can be installed with the command \texttt{apt-get install hunor}. Ispell has a \TeX\ mode, which skips control sequences and comments when checking \TeX\ source. (Unfortunately, the arguments of \texttt{\begin{tabular}} and many other non-textual elements of \TeX\ documents are not skipped.) Ispell can be used interactively, but this method is not comfortable, and incremental checking is not possible.

Ispell also has an interprocess communication protocol, through which it can be integrated into text editors. For example, Emacs has built-in Ispell support to mark incorrect words visually. OpenOffice, LyX, editors in KDE and newer versions of Vim can do the same. MagyarSpell works fine in these editors. It is not part of Magyar\TeX, but it is freely available.

Note, however, that both the database and the stemmer of MagyarSpell is far from perfect, but among the Hungarian spell checkers only this one works inside Ispell, so only this can be easily integrated into editors.

MSpell A commercial Hungarian spell checker with a no-cost Linux download, developed by Morphologic (a company in Hungary that produces linguistic software). Doesn’t have an interactive mode, but can replace Ispell in inter-process communication mode. A shell script is provided that replaces the ispell command, so MSpell can be integrated into text editors more easily. It is not part of Magyar\TeX.

HunSpell The successor of MagyarSpell, but based on a different spell checking architecture. It understands Hungarian much better than MagyarSpell, but since it is not based on Ispell, it is harder to integrate into text editors. For example, it is not available from the Emacs spell checking menu, even if it is installed. It is not part of Magyar\TeX, but it is freely available.

4.2 Future Work

Some features are still missing from magyar.lfd:

- \texttt{letter.cls} is not customized properly, the left indentation of the nested list environments is also not customized;
- a macro to emit numbers with groups of three digits separated is missing;
- layout.sty and many other packages don’t have Hungarian captions yet;
- the shorthand ‘\texttt{\textdagger}’ is not disabled in math mode to give nath.sty a chance to typeset $H_{\text{symm}}$ with $\texttt{\textdagger\{\text{symm}\}}$;
- \texttt{\hunnewlabel} should store \texttt{table}, \texttt{figure} or \texttt{equation}, so \texttt{\refstruct} can insert it;
- new fonts and/or methods should be developed in place of ‘tty';
Hungarian typography needs a baseline grid, which is almost impossible to enforce in \LaTeX;

• some of the separation symbols proposed for after `\paragraph` are not available yet;

• section titles should not be larger than normal text;

• bold fonts should be substituted for bold extended fonts, whenever available—and never with an error or warning message; page numbers should be removed from blank pages;

• the width of `\parindent` should be computed based on `\textwidth`;

• the length of the last line of a paragraph should not be too near to the right margin, especially if `\parindent = 0`;

• `\vskip`s above and below sections should be reduced;

• providing the commands `\H` and `\.` for OT1-encoded typewriter fonts;

• `\MathReal` should be extended with physical units;

• virtual fonts to support `\umlautlow` in T1 encoding;

• bold in `\begin{description}` should be `\textsc`;

• allow specifying some compile-time options at run-time;

• `\textqq` should also work as an environment;

• `\told` should generate suffixes for month names.

Other features should be implemented outside `magyar.ldf`, as external programs. All programs in section 4.1 need improvement in one way or another.

4.3 Conclusion

An updated `magyar.ldf` which closely follows Hungarian typographical rules and works together with the most popular \LaTeX packages without problems, has been awaited for many years. This new version is ready, as a single file longer than anything before, and it is filled with many advanced features.

Most of the features adapt \LaTeX to Hungarian typographical rules, but some of them are bug fixes to various external packages, including design flaws and compatibility issues in \Babel itself.

The implementation of some features clearly shows that \TeX macro programming is an obscure and ineffective way of solving some of the language-related problems. It is hoped that new versions of \Omega, together with the new version of \Babel, will provide a framework in which such problems can be addressed compactly and elegantly, without constant awareness of actual and possible compatibility glitches.

References


