Typesetting multilingual documents with ANTOMEGA *

Alexej Kryukov

May 8, 2005

Abstract

Antomega is a language support package for Lambda, based on the original `omega.sty` file. However, it provides some additional functionality.

1 Introduction

Moving from LaTeX to Ω is always difficult for an average user, since the Ω distribution doesn't include any language support package which could be used as a Babel replacement. The `omega.sty` file, version of 1999/06/01, was released by the Ω developers as a first attempt to make something like 'Omega-Babel', but, unfortunately, this work was not finished. Moreover, more recent versions of `omega.sty` are suitable only for testing some right-to-left languages, but not for regular work. So I prepared my own package, based on the original `omega.sty`, which fixes some bugs and provides some additional functionality.

2 Installation instructions

First, download and install the Ω binaries or ensure that your TeX installation already includes them. Unpack the archive file with ANTOMEGA and move all files to the appropriate directories (for example, everything in `omega/lambda/` to `$$texmf/omega/lambda`, everything in `omega/ocp/` to `$$texmf/omega/ocp`, and so on. If you already have a file named `language.dat` in `$$texmf/omega/lambda/base`, replace it with the provided file (called language.dat.sample) in case you want to get correct hyphenation for Russian and/or Greek.

Note that ANTOMEGA still needs some files from the original Ω distribution. The most important file is `ut1omlgc.fd`. Unfortunately, this file was not included into some recent Ω distributions. I can neither include it into my package as is (this might cause name clashes) nor rename it (since I can't rename the default font and the default encoding vector used in Ω). So in case you haven't this file already installed you have to install it separately. Either take it from an older TeX distribution or from the Ω CVS tree.

There are also some additional translation processes (useful mainly for typesetting polytonic (classical) Greek), which you may want to take from old Ω.

*This file has version number 0.8, last revised on 7 May 2005.

Of course, after installing new files you have to update the TEX file names database. On teTeX or fpTeX systems this is performed with `texconfig rehash` or `mktexlsr` commands. On MikTeX you can do the same via a menu item. Refer yourself to a special section (8) of this manual, in case you are interested in installing and configuring ANTOMEGA under Scientific Word/Scientific Workplace.

2.1 Deprecated files

If you are upgrading from an older version of ANTOMEGA, you can safely delete the following deprecated files:

- In `$$TEXMF/omega/unidata`:
  - `uni0300.def` (now called `uni0370.def`;
  - `grahyph.tex` and `grmhyph.tex` (replaced with `ograhyph.tex` and `ogrmhyph.tex` correspondingly according to the naming convention proposed by Dimitrios Filippou);
- In `$$TEXMF/omega/hyphen`: `greek2uni.tex` and `greek2omega.tex` (no longer needed);
- In `$$TEXMF/omega/otp/antomega`: `latin2punct.otp` and `cyr2punct.otp` (replaced with files `tex2punct.otp`, `babel2punct.otp` and `babel2ru.otp`);
- In `$$TEXMF/omega/ocp/antomega`: `latin2punct.ocp` and `cyr2punct.ocp` (replaced with files `tex2punct.ocp`, `babel2punct.ocp` and `babel2ru.ocp`).

2.2 Updating Lambda format

With older versions of ANTOMEGA you could do without updating Lambda format (at least if you work only with Latin-based languages), but now this operation is strongly recommended for the following reasons. As you probably know, the core of the LATEX system consists of `latex.ltx` and some other files which should be loaded into format. Since there are no special versions of those files for Ω, the same files are used also for Lambda format.

Of course, files designed for LATEX not always work well for Ω. For example, in order to get correct hyphenation for a text in a specific language, we have to set `\catcode` (which should be equal to 11 or 12) and `\lccode` for all characters used in that language. For correct conversion to uppercase we need also `\uccode` values. In standard LATEX these values are defined for all 256 characters of the encoding table, but with Ω we need similar definitions for a wider range of Unicode characters, and it would be nice to have these definitions loaded into format. Some people achieve this effect putting tables of such codes together with their hyphenation patterns.

However, this approach causes some problems. First of all, it is very inconvenient to set `\catcode`, `\lccode` and `\uccode` with primitive commands for each letter from a large amount of characters. Antomega defines some commands (namely `\makeletter`, `\makeucletter`, `\makelcletter` and `\makesameletter`), allowing to simplify this process. It also provides definition tables for some Unicode ranges, written using these commands. However, in order to load these tables into the format the commands they use should be already known to `iniomega`.

There is another problem, even more important. The file responsible for loading hyphenation patterns is called `hyphen.cfg`. This file is part of the `babel` package, but some language-specific formats like cslatex, platex etc. include their own versions of `hyphen.cfg`. Note that those versions are mainly incompatible with ANTOMEGA. The official Ω distribution also included its own simplified version of `hyphen.cfg`, but this file was removed from the most recent versions. That's why original babel's version of the file is also often loaded into Lambda format.

This `hyphen.cfg` is mainly compatible with Ω, although it defines a lot of specific commands not needed if we are not planning to use Babel itself. However, in last versions it has the following feature: hyphenation patterns are loaded inside a TEX group. This means that hyphenation rules itself will be saved, but character codes loaded together with them will be forgotten immediately after processing hyphenation rules. That's correct, because character codes required by specific hyphenation patterns may not match codes normally used by LATEX. For example, Russian hyphenation patterns usually have the koi8 encoding, which is not directly used by LATEX. So we need a specific table of codes in order to reencode these patterns into an internal font encoding supported by TEX. Once reencoding is performed, these character codes are no longer needed.

However, this means that hyphenation patterns is an incorrect place to store character codes for Unicode symbols, because it is very possible that our settings will take no effect. And even if they are saved (in case we have old Omega's `hyphen.cfg` installed) the result may be rather unexpected, because Ω has no way to determine which codes are necessary only for processing specific hyphenation patterns, and which should really be stored for further use.

That's why ANTOMEGA now includes its own version of `hyphen.cfg` and special file `antomega.cfg` which contains references to tables of character codes for all supported Unicode ranges. This version of `hyphen.cfg` first defines commands `\makeletter`, `\makelcletter`, `\makeucletter` and `\makesameletter`. After that it loads `antomega.cfg`. So if you created a custom table of character codes for your script, you may load it via `antomega.cfg` instead of including it into your hyphenation patterns. Of course you can also prevent codes for a specific Unicode range from loading into your format. For example, if you never use polytonic Greek, comment out the following line in `antomega.cfg`:

```
\input{uni1f00.def}
```

Only after loading character codes into format, hyphenation patterns are processed. As well as in Babel, this procession is done inside a group, and so all character codes defined here should be used only for converting your patterns into another encoding.

Note that, while old Omega's `hyphen.cfg` resides in `$TEXMF/omega/lambda/config`, ANTOMEGA installes its version of the file into `$TEXMF/omega/lambda/antomega`. This prevents name clashes, but, in case you have old `hyphen.cfg` installed, you have to remove it manually to ensure that the correct version will be found and loaded by `iniomega`. Only after that you may rebuild the `lambda` format file. On teTeX or fpTeX systems you have to run

```
fmtutil --byfmt lambda
```

See section 8 for information on doing that with TrueTEX/ Scientific Word systems.

It is easy to test if you have the correct `hyphen.cfg` version loaded into your format: just take a look at any `.log` file produced by `lambda`. In case everything is correct it should contain the following text at the beginning: "Antomega and hyphenation patters for... loaded". Now you should be able to typeset you documents with ANTOMEGA.

3 Loading ANTOMEGA

One of the main advantages of `omega.sty` was using different commands for setting the main language of the document and for loading additional languages. ANTOMEGA preserves this feature, using the same `\background` and `\load` commands. So if you want to prepare an English document including some Greek text, you can do it by the same way as with `omega.sty`, for example:

```
\usepackage{antomega}
\background{english}
\load{greek}
```

However, `omega.sty` needs two different files for each language: first of them (with the *.bgd extension) is used by the `\background` command, and second (with the *.lay extension) by the `\load` command. Of course, these two files usually have very similar code. ANTOMEGA fixes this problem: both `\background` and `\load` commands load the same language definition file with the .ldf extension, but process it in a different way.

4 Typesetting in different languages

`omega.sty` supported only a limited set of languages, which included `usenglish`, `french` and `greek`. ANTOMEGA supports these languages too, but separate support for `usenglish` is no longer available. Instead you can load `english` with options `dialect=british` or `dialect=american`, for example:

```
\background[dialect=american]{english}
```

I added support files for Russian, and later also for German, Polish and Latvian.

Generally speaking, it is not difficult to provide support for a new language, since language definition files are quite independent from the core package, and so you can write a file with definitions for your language without changing anything in `antomega.sty`, using the existing .ldf files as an example.

In the original `omega` package we could use for switching to another language either an environment with the same name as a name of your language, or (for small pieces of text) the `\local<$language>` macro, there `<$language>` is your language name. These commands had to be defined in the language definition file. For example, `usenglish.bgd` defined the `usenglish` environment and the `\localusenglish` command.

These commands are still supported in ANTOMEGA. However, beginning from the version 0.6 ANTOMEGA provides new language switching commands, compatible with the Babel package. So you can use the `\selectlanguage` and `\foreignlanguage` macros and the `otherlanguage` environment exactly as you did with Babel. This means that your old documents may be transferred to Ω with minimal changes.

5 Loading languages with options

As well as the original `omega.sty` file, ANTOMEGA requires the `keyval` package. So all commands used for loading languages may be executed with different parameters, which may take different values. Each language has its own set of such parameters. However, some options are suitable for all supported languages. The most important of them are `input` and `output` parameters, which replace the `inputenc` and `fontenc` packages, used in standard LaTeX.

Beginning from ANTOMEGA v. 0.7 the same `keyval` syntax is also supported for options of the `antomega` package itself.

5.1 The "input" parameter

Of course, this parameter is language-specific. However, there are two values, which are always supported: `utf-8` and `ucs-2`. The later really means "no conversion", since ucs-2 is the native format for Ω. Since these two encodings are suitable for most languages, they may be specified in options of the ANTOMEGA package, for example:

```
\usepackage[input=utf-8]{antomega}
```

The `\load` and `\background` commands also support this option, but they may additionally accept other values for it, depending from the language you want to load. For example, if you want to type an English document with some international symbols encoded in iso-8859-1, you may put the following line into your LaTeX preamble:

```
\background[input=iso-8859-1]{english}
```

5.2 The "output" parameter

For this parameter you can use one of the following values: `unicode`, `omega` and `tex`. `unicode` is used by default. Note that the `omlgc` font, distributed with Ω, is not fully compatible with Unicode. For example, it has a specific encoding for Latin ligatures and general punctuation. So you have to set `output=omega` if you want to use this font, and `output=unicode` if you have another font, more strictly conforming the Unicode standard. You my want to set also `output=tex` if you prefer using 8-bit fonts in standard TeX encodings (T1 for Western languages, T2A for Russian, LGR for Greek).

For example, if you want to typeset your English text with the standard EC fonts, but haven't any corresponding font for Greek, you may use the following preamble:

```
\documentclass{article}
\usepackage{antomega}
\background[output=tex]{english}
\load[output=omega]{greek}
```

5.3 The "shorthands" parameter

Since ANTOMEGA tries to completely reproduce the functionality of the Babel package, it also supports combinations with the " character (*shorthands*) which have a special meaning in Babel. Note that ANTOMEGA always uses translation processes to emulate Babel's behavior, and so it never really makes " an active character. The set of supported shorthands differs from language to language, but there is a minimal set available by default. For example, you may use "< and "> for guillemots. You may turn off support for Babel shorthands by setting 'shorthands=off', and, of course, you may explicitly enable it ('shortahnds=on'). Currently this parameter is supported for all languages except Greek.

6 Translation processes

Since the last Ω versions are suitable only for testing purposes, they don't include many useful files, originally provided by J. Plaice ang Y. Haralambous. Particularly some Ω translation processes were removed, and some are incorrect (e. g. don't correspond to the `omlgc` font). That's why `antomega` provides its own set of `ocp` and `otp` files, which makes it rather independent from Ω's texmf part.

For conversion to different encodings I added some new .otp and .ocp files, which (I hope) work correctly. Beginning from v. 0.6 ANTOMEGA includes improved translation processes for conversion from some standard iso-8859 and windows-125* codepages to Unicode. However, some original `.ocp` files are still necessary for `antomega` to work. There also some rarely used (but still supported in `antomega`) files, not present neither in `antomega` nor in the most recent Ω distributions.

7 Selecting fonts with ANTOMEGA

Of course, it is not enough to set an input encoding for your language. You will need also a correct font matching your encoding. With ANTOMEGA you can select a font separately for each script you use. For example, it defines new commands `\westernrm`, `\westernsf` and `\westerntt`. So, if you want to use Computer Modern for English but prefer to keep standard omlgc for Greek, simply put the following line in your preamble:

```
\renewcommand{\westernrm}{cmr}
```

In ANTOMEGA's language support files some similar commands for other scripts are defined. For example, `omega-russian.ldf` introduces `\russianrm`, and `omega-latvian.ldf` — `\balticrm`. Note that there is no need to define specific commands for Central European languages, because not only Unicode, but even T1 covers both Western and Central European character sets.

Of course, you can write your own special packages to make selecting a new font a bit more easy. You can even use standard font selecting packages, but, in this case, you must load them *after* ANTOMEGA itself and *before* any language-specific commands. For example:

```
\usepackage{antomega}
\usepackage{palatino}
\background{english}
```

8 Using ANTOMEGA with MacKichan software products

Generally speaking, adapting MacKichan software products (like Scientific Word or Scientific Workplace) for non-Latin languages represents a not so trivial task. The problem is, that their shell (very powerful by itself) knows nothing about various input encodings, supported by LaTeX, and so it can't take advantage of multilingual capabilities, provided by Babel. Instead, it represents any national characters, typed by user, in the form `\U{<hexadecimal Unicode index>}`. This representation is hardly legible for standard LaTeX, because it is Unicode-based. That's why Ω is traditionally used for preparing multilingual documents with Scientific Word or Scientific Workplace.

So, adapting ANTOMEGA (which is already distributed together with these software packages) to MacKichan shell is a very natural solution for those who want to get their text correctly typeset according to typographic rules used in their native languages. Preparing your Scientific products to use ANTOMEGA may be divided into two main stages: configuring your TrueTeX installation (which lies in the background of Scientific Word/Scientific Workplace) and configuring the graphical front-end itself.

8.1 Configuring TrueTeX

First, ensure you have ANTOMEGA v. 0.8 or above. If necessary, download the latest version from CTAN and copy the contents of all directories available in the downloaded package into the corresponding directories found in `TCITeX\Omega\`.

Second, locate the file `TCITeX\Omega\Lambda\base\languages.dat.sample` and rename it to `languages.dat`. Edit this file to enable hyphenation patterns for your language.

Now you have to rebuild the Lambda format. This operation is mandatory, because all default format files supplied with Scientific products are built without multilingual extensions provided by Babel, so that using ANTOMEGA with your default `lambda.oft` just will cause an error. This is also probably the hardest part, because TrueTeX, unlike MikTeX or fpTeX, provides no special tools allowing to call `iniomega` with desired parameters. However, to simplify this task, you can use the following batch script (call it, say, `runinilambda.bat`):

```
echo off
setlocal
set TEXMF=C:\sw50\TCITeX
set TEXINPUTS=.;%TEXMF%/{omega,tex}//
```

```
%TEXMF%\web2c\iniomega %TEXMF%\omega\lambda\config\lambda.ini
endlocal
```

Note that you can run this script from any location, but the resulting `lambda.oft` file must be placed into your `TCITeX\web2c\` directory.

8.2 Configuring the Scientific Word / Scientific Workplace shell

When you have done with updating Lambda format, you can start Scientific Word and create a new document (or load an existing one). To make its processing with Lambda possible, you should do the following:

- From the main menu bar, select 'Typeset'—'Expert Settings'. The dialog box with several tabs will appear, where you should select the 'DVI Format Settings' page, and then the 'TeX Live Lambda' entry from the drop-down box.
- Go to the 'Typeset'—'Options and packages' menu. Ensure that `sw2unicode`, `swtimes` and `fontenc` are **not** in the list of loaded packages. These packages are not needed, since their functuionality is completely incorporated by ANTOMEGA.
- From the same dialog box, click the 'Go native' button. A dialog box with a multi-line input field will appear. Add the following line to that input field:

  ```
  [input=sw,ffi=ligatures]{antomega}
  ```

  Note that the `ffi=ligatures` option is mandatory, since Times New Roman (the only Unicode font which is supported in the Scientific Word/Scientific Workplace distribution by default) hasn't some latin ligatures at the places ANTOMEGA expects to find them, so that enabling the corresponding translation process will result in missing glyphs in the output. Unfortunately, the default `ofm` files for Times New Roman also contain no information about ligature substitution, so that `ffi=ligatures` practically means 'no ligatures at all' in this case.
- Go to 'Typeset'—'Preamble' and input any number of `\background` and `\load` commands for the languages you are planning to use.

Now you should be able to compile your document with Lambda.

9 The ANTOMEGA code

9.1 Handling identification codes for our files

Unlike in Babel, no tests if `\ProvidesFile` is already defined. Since LaTeX 2_ε was released in 1994, and the Ω project started also in 1994, probably nobody will use Ω with the 2.09 format.

\ProvidesFile

We save the original definition of \ProvidesFile in \ant@tempa and restore it after we have stored the version of the file in \toks8.

```
\let\ant@tempa\ProvidesFile
\def\ProvidesFile#1[#2 #3 #4]{%
   \toks8{Antomega <#3> and hyphenation patterns for }%
   \ant@tempa#1[#2 #3 #4]%
   \let\ProvidesFile\ant@tempa}
```

\ProvidesLanguage

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

```
\def\ProvidesLanguage#1{%
   \begingroup
      \catcode`\ 10 %
      \@makeother\/%
      \@ifnextchar[%]
         {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
\def\@provideslanguage#1[#2]{%
   \wlog{Language: #1 #2}%
      \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
      \endgroup}
\ProvidesFile{hyphen.cfg}
   [2005/05/07 v0.8
   Taken from Babel language switching mechanism
   and modified for Antomega]
```

9.2 Handling ΩCP files

\LoadOCPByName

The macro \LoadOCPByName takes two arguments: an OCP file name (without extension) and an Ω command which will be used for loading this file. If the referenced .ocp file doesn't exist in user's system, `id.ocp` will be used instead. So it is possible to proceed with document processing, even if some .ocp files were not found.

```
\def\LoadOCPByName#1#2{\IfFileExists{#2.ocp}{\ocp#1=#2}{
   \PackageWarning{antomega}{#2.ocp not found.
      Identity will be used instead.}{}
   \ocp#1=id}}
```

Now we load some commonly used translation processes, using the macro \LoadOCPByName.

```
\ocp\IdOCP=id
\LoadOCPByName{\BasicIsoUni}{uniutf2uni}
\LoadOCPByName{\BasicWinUni}{uniutf2uni}
\LoadOCPByName{\BasicUtfUni}{uniutf2uni}
\LoadOCPByName{\BasicTexUni}{tex2punct}
\LoadOCPByName{\BasicBabelUni}{babel2punct}
\LoadOCPByName{\BasicAccentsUni}{uni2accents}
\LoadOCPByName{\UniToOmega}{uni2omega}
\LoadOCPByName{\Oldstyle}{oldstyle}
\LoadOCPByName{\LatinUniToTex}{uni2t1}
```

`uni2lig.ocp` is used for setting up Latin ligatures. However, the standard TEX ligature mechanism should be a better choice, since using OCP for combinations like 'fi' or 'fl' may break hyphenation. So I provided a special `ffi` option which

may be set either to 'ocp' or to 'ligatures'. Setting it to 'ligatures' simply prevents the translation process from loading. Note that `uni2lig.ocp` is designed for pure Unicode fonts and it is never used, if `output` is set to 'omega'. In this case you can't turn off processing ligatures via OCP, since the `omlgc` font doesn't contain any ligatures at all.

```
\def\opt@ocp{ocp}
\def\opt@ligatures{ligatures}
\define@key{antomega}{ffi}[ocp]{%
   \def\@tmpa{#1}
   \ifx\@tmpa\opt@ocp%
      \LoadOCPByName{\LatinUniToLig}{uni2lig}
   \else\ifx\@tmpa\opt@ligatures%
      \typeout{Antomega package option: ffi=ligatures}
      \LoadOCPByName{\LatinUniToLig}{id}
   \fi\fi
}
```

The following option is deprecated and preserved for backwards compatibility only. Use 'ffi=ligatures' instead.

```
\DeclareOption{noffi}{\setkeys{antomega}{ffi=ligatures}}
```

By default we use OCP for Latin ligatures in order to prevent MikTeX crashes.

```
\setkeys{antomega}{ffi=ocp}
```

Now we can define some standard OCP lists, useful generally for languages with Latin-based scripts.

`\BasicTexOCP` This OCP list loads the default translation process for standard TeX ligatures and punctuation characters.

```
\ocplist\BasicTexOCP=
   \addbeforeocplist 1750 \BasicTexUni
\nullocplist
```

`\BasicBabelOCP` This OCP list loads the default translation process for Babel-like shorthands. If you don't like them, set 'shorthands=off' for your language.

```
\ocplist\BasicBabelOCP=
   \addbeforeocplist 2000 \BasicBabelUni
\nullocplist
```

`\BasicAccentsOCP` This OCP list converts Unicode combining accents to TEX-styled `\accent` commands.

```
\ocplist\BasicAccentsOCP=
   \addbeforeocplist 2250 \BasicAccentsUni
\nullocplist
```

`\BasicInputUcsOCP` This is ANTOMEGA's default OCP list. It doesn't translate text to any other character set (so, sctually, does nothing).

```
\ocplist\BasicInputUcsOCP=
   \addbeforeocplist 500 \IdOCP
\nullocplist
```

`\BasicInputUtfOCP` This OCP list should be used for utf-8 encoded texts.

```
\ocplist\BasicInputUtfOCP=
   \addbeforeocplist 500 \BasicUtfUni
\nullocplist
```

\BasicInputIsoOCP This OCP list is intended for 8-bit texts using Latin ISO-8859-1 codepage, but note that it doesn't perform any real conversion, since ISO-8859-1 character codes are the same as in Unicode, and Ω automatically distinguishes 8-bit and 2-byte input.

```
\ocplist\BasicInputIsoOCP=
   \addbeforeocplist 500 \BasicIsoUni
\nullocplist
```

\BasicInputWinOCP This OCP list is intended for 8-bit texts using Latin windows-1252 codepage.

```
\ocplist\BasicInputWinOCP=
   \addbeforeocplist 500 \BasicWinUni
\nullocplist
```

The following OCP lists are used to convert a text to an Ω output.

\LatinUniOutOCP A conversion to a Unicode font. The only operation which may be performed here is setting up the Latin ligatures.

```
\ocplist\LatinUniOutOCP=
   \addbeforeocplist 3500 \LatinUniToLig
\nullocplist
```

\LatinOmegaOutOCP A conversion to the default omlgc font. Its encoding differs from Unicode, and so a special conversion routine is required.

```
\ocplist\LatinOmegaOutOCP=
   \addbeforeocplist 3500 \UniToOmega
\nullocplist
```

\LatinTexOutOCP A conversion from Unicode to the T1 encoding.

```
\ocplist\LatinTexOutOCP=
   \addbeforeocplist 3500 \LatinUniToTex
\nullocplist
```

\OldstyleOCP This OCP list converts ASCII digits to their oldstyle equivalents. Note that it is not compatible with the `omlgc` font.

```
\ocplist\OldstyleOCP=
   \addbeforeocplist 4000 \Oldstyle
\nullocplist
```

The following key allows to set input encoding globally for the whole document instead of setting it separately for each language. Of course, from all the standard codepages only 'utf-8' makes a sense in this context. 'ucs-2' is also supported, but this encoding doesn't require any translation processes, because Ω uses it by default anyway.

Beginning from ANTOMEGA v. 0.8 you can also select 'sw' to match a specific Unicode character representation, used in files generated by MacKichan software products.

```
\let\BasicInputOCP\BasicInputUcsOCP
   \define@key{antomega}{input}[ucs-2]{
      \def\@tmpa{#1}%
      \ifx\@tmpa\opt@utf%
         \let\BasicInputOCP\BasicInputUtfOCP%
         \typeout{Antomega package option: input=utf-8}
      \else\ifx\@tmpa\opt@sw%
         \def\U##1{/QQ[##1]}%
```

```
88          \def\rmdefault{swtimes}%
89          \let\westernrm\rmdefault%
90          \LoadOCPByName{\BasicSWordUni}{sw2uni}%
91          \ocplist\BasicInputSWordOCP=
92             \addbeforeocplist 500 \BasicSWordUni
93          \nullocplist
94          \let\BasicInputOCP\BasicInputSWordOCP%
95          \typeout{Antomega package option: input=sw}
96       \else%
97          \let\BasicInputOCP\BasicInputUcsOCP%
98          \typeout{Antomega package option: input=ucs-2}
99       \fi}
```

`\UppercaseOCP` . ANTOMEGA includes a special translation process, `uppercase-dflt.ocp`, based on the `uppercase.ocp` file, available in older Ω distribution, which may be used for lowercase to uppercase conversion. Although standard conversion rules, based on `\lccode` and `\uccode` settings, usually produce a better result, I have to use the OCP-based conversion by default, because Ω incorrectly processes some character codes in its UTF-8 mode. You can disable this feature by setting `uppercase=standard`.

```
100 \LoadOCPByName{\Uppercase}{uppercase-dflt}
101 \ocplist\UppercaseOCP=
102    \addbeforeocplist 3000 \Uppercase
103 \nullocplist
104 \def\MakeUppercase#1{{\pushocplist\UppercaseOCP#1}}
105 \def\opt@standard{standard}
106 \define@key{antomega}{uppercase}[ocp]{
107    \def\@tmpa{#1}
108    \ifx\@tmpa\opt@standard
109       \let\MakeUppercase\uppercase
110       \typeout{Antomega package option: use character codes}
111       \typeout{for conversion to Uppercase}
112    \fi}
```

`\oldstylenums` This command supposes that our text font contains old style numerals and that they are mapped to their places in the Unicode Private Use area as defined in AGL. Don't use it with the `omlgc` font.

```
113 \def\oldstylenums#1{{\pushocplist\OldstyleOCP#1}}
```

9.3 Encoding-independent commands for printing special characters

`\noocpchar` After expanding a command, Omega puts the result back into the OCP stack. If the result of expansion contains some characters, which have a special meaning in the TEX system, they will be processed again, instead of typing into the output. So it is impossible e. g. to use the `\%` command in order to obtain the percent sign, because the character produced by that command is recognized anyway as a comment mark after processing it via OCP. The same problem affects character codes less than 0x20 (used in most 8-bit encodings), since these characters usually have no `\catcode` assigned, and so are treated as invalid in the input.

Previously some hacks were used to prevent this effect. For example, in the `omlgc` some ASCII characters are reproduced once again in the 0x80–0x0F range,

not used in Unicode, so that e. g. the \% command could actually refer to a slot different from the 'real' percent sign. For 8-bit fonts we had to assign code 12 to some characters in order to make them 'valid'. Older ANTOMEGA versions had a special option, specials, used to control such situations.

However, this option is deprecated (and removed) now. Instead, we just define a special command which puts \clearocplists before \char in order to prevent the result from placing into the OCP stack...

```
\def\noocpchar#1{{\clearocplists\char#1}}
```

... and then apply it to the most commonly used special characters.

```
\def\#{\noocpchar{"23}}
\def\%{\noocpchar{"25}}
\def\&{\noocpchar{"26}}
```

9.4 Omega-specific commands to handle TEX font encodings

\uniencoding

It is necessary to declare a special encoding for Omega-specific 2-byte fonts. Ω developers called it UT1.

```
\def\uniencoding{UT1}
```

\ant@load@encoding

The concept of current font encoding doesn't really matter for ANTOMEGA. Although some text commands traditionally used in LATEX are allowed in the input, they just should be always mapped to the same Unicode codepoints, and it is a task of translation processes to translate them to the current font encoding.

So the only reason why we have to declare font encodings at all is that Omega needs to know the current encoding in order to select an appropriate font. That's why antomega no longer loads any encoding definition files. Instead, whatever encoding is requested, antomega always declares it itself, and then loads the same list of mappings between text commands and Unicode codepoints.

```
\def\ant@load@encoding#1{%
   \edef\ant@encodingfile{%
      \lowercase{\def\noexpand\ant@encodingfile{#1enc-antomega.def}}}%
   \ant@encodingfile
   \InputIfFileExists{\ant@encodingfile}{}{%
      \DeclareFontEncoding{#1}{}{}
      \PackageWarning{antomega}{The \ant@encodingfile\ file was not found.
         The #1 encoding was defined by antomega.}{}
      }
   \let\ant@encodingfile\@undefined
}
\ant@load@encoding{\uniencoding}
\def\encodingdefault{\uniencoding}
```

Since T1 is loaded into the Lambda format, we have to redefine it now.

```
\ant@load@encoding{T1}
```

9.5 Font issues

The omlgc font is not perfect, but it is included into all standard TEX distributions. So, it will be used by default.

```
\def\rmdefault{omlgc}
```

`Antomega` stores its default font names in the `\westernrm`, `\westernsf` and `\westerntt` variables, since `\rmdefault`, `\sfdefault` and `\ttdefault` will be redefined each time we switch to a new language.

```
\ifx\westernrm\@undefined\let\westernrm=\rmdefault\fi
\ifx\westernsf\@undefined\let\westernsf=\sfdefault\fi
\ifx\westerntt\@undefined\let\westerntt=\ttdefault\fi
```

Generally speaking, with Ω we should use translation processes rather than active characters. So I made `textasciitilde` an 'other symbol'.

```
\catcode`\~=12
```

9.6 Language-specific commands which should be loaded into the Lambda format

Again, no tests for `\language` and `\newlanguage`, because Ω should be always compatible with TEX version 3.0.

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

`\addlanguage` To add languages to TEX's memory plain TEX version 3.0 supplies `\newlanguage`. However, a new macro is defined here, because the original `\newlanguage` was defined to be `\outer`.

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

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

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

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

```
\def\iflanguage#1{%
   \expandafter\ifx\csname l@#1\endcsname\relax
      \PackageWarning{antomega}{#1 is not a known language.}%
   \else
      \ifnum\csname l@#1\endcsname=\language
         \expandafter\@firstoftwo
      \else
         \expandafter\@secondoftwo
      \fi%
   \fi}
```

9.7 Handling character codes

We can't get correct hyphenation for our 2-byte characters without setting `\catcode`, `\lccode` and `\uccode` for each of them. The following commands simplify making such definitions.

`\makeletter` This command takes two arguments, the first being an uppercase character and

the second a corresponding lowercase character, and sets \lccode and \uccode for both characters.

```
\ifx\makeletter\@undefined
   \def\makeletter#1#2{%
      \ifnum\catcode#2=11\else\catcode#2=12 \fi
      \ifnum\catcode#1=11\else\catcode#1=12 \fi
      \uccode#1=#1 \uccode#2=#1%
      \lccode#1=#2 \lccode#2=#2}
\fi
```

\makelcletter This command takes two arguments, the first being an uppercase character and the second a corresponding lowercase character, and sets \lccode and \uccode for the lowercase character.

```
\ifx\makelcletter\@undefined
   \def\makelcletter#1#2{%
      \ifnum\catcode#2=11\else\catcode#2=12 \fi
      \uccode#2=#1%
      \lccode#2=#2}
\fi
```

\makeucletter This command takes two arguments, the first being an uppercase character and the second a corresponding lowercase character, and sets \lccode and \uccode for the uppercase character.

```
\ifx\makeucletter\@undefined
   \def\makeucletter#1#2{%
      \ifnum\catcode#1=11\else\catcode#1=12 \fi
      \uccode#1=#1%
      \lccode#1=#2}
\fi
```

\makesameletter This command takes two arguments, both of them being uppercase or lowercase characters, and sets \lccode and \uccode for character 1 equal to character 2.

```
\ifx\makesameletter\@undefined
   \def\makesameletter#1#2{%
      \ifnum\catcode#1=11\else\catcode#1=12 \fi
      \uccode#1=\uccode#2%
      \lccode#1=\lccode#2}
\fi
```

The following code should be written into antomega.cfg. You may edit that file depending from which Unicode ranges you really need.

```
\input{uni0100.def} % Latin Extended-A
\input{uni0370.def} % Greek Basic
\input{uni0400.def} % Cyrillic
\input{uni1f00.def} % Greek Extended
```

In hyphen.cfg first we test if antomega.cfg exists, and then load it.

```
\openin1 = antomega.cfg
\ifeof1
   \message{I couldn't find the file antomega.cfg.\space
            Codes for Unicode characters will not be loaded.}
\else
   \input{antomega.cfg}
\fi
\closein1
```

9.8 Warnings and error messages

\ant@nocodes This command is used to show a warning message if Ω can't find a file with lccodes/uccodes for the specified Unicode range.

```
\providecommand*{\ant@nocodes}[3]{%
   \PackageWarningNoLine{antomega}%
      {No file was found with symbol codes\MessageBreak
         for the #2 range #3.\MessageBreak
         You may proceed, but your #1 texts\MessageBreak
         probably will not be correctly hyphenated.}}
```

\ant@nopatterns This macro is based on Babel's `\@nopatterns` command. However I removed test if `\PackageWarningNoLine` is defined, because probably nobody will try to build LaTeX 2.09 based format for Ω.

```
\providecommand*{\ant@nopatterns}[1]{%
   \PackageWarningNoLine{antomega}%
      {No hyphenation patterns were loaded for\MessageBreak
         the language `#1'\MessageBreak
         I will use the patterns loaded for \string\language=0
         instead}}
```

\ant@nolang This macro defines the error message which will be displayed if the requested language definition file was not found.

```
\providecommand*{\ant@nolang}[1]{%
   \PackageWarningNoLine{antomega}%
      {Couldn't find file omega-#1.ldf!!}}
```

9.9 Different corrections for standard LaTeX commands

With Ω we usually have to control all commands which print some strings (for example, to headers/footers), so that they always apply correct translation processes and correct font to the text they produce. However, modifying these commands may be inconvenient if we have to use some packages which also try to redefine them. If you want to prevent `antomega` from modifying these commands, you may control its behavior by setting the `localmarks` option either to 'on' or to 'off'.

\local@marks This command is executed every time we are switching to a new language. It applies all rules specific for this language to the text, which is written to headers/footers.

```
\def\opt@enabled{on}
\def\opt@disabled{off}
\def\opt@tex{tex}
\def\opt@omega{omega}
\def\opt@unicode{unicode}
\def\opt@utf{utf-8}
\def\opt@ucs{ucs-2}
\def\opt@sw{sw}
\define@key{antomega}{localmarks}[on]{%
   \def\@tmpa{#1}
   \ifx\@tmpa\opt@enabled
      \def\local@marks##1{%
         \def\markboth####1####2{%
            \begingroup%
```

```
                \let\label\relax \let\index\relax \let\glossary\relax%
                \unrestored@protected@xdef\@themark%
                {{\foreignlanguage{##1}{####1}}{\foreignlanguage{##1}{####2}}}%
                \@temptokena \expandafter{\@themark}%
                \mark{\the\@temptokena}%
             \endgroup%
             \if@nobreak\ifvmode\nobreak\fi\fi}%
             \def\markright####1{%
                \begingroup%
                   \let\label\relax \let\index\relax \let\glossary\relax%
                   \expandafter\@markright\@themark{\foreignlanguage{##1}{####1}}%
                   \@temptokena \expandafter{\@themark}%
                   \mark{\the\@temptokena}%
                \endgroup%
                \if@nobreak\ifvmode\nobreak\fi\fi}%
             \def\@markright####1####2####3{\@temptokena{####1}%
                \unrestored@protected@xdef\@themark{{\the\@temptokena}%
                {{####3}}}}}
   \else\ifx\@tmpa\opt@disabled
      \def\local@marks#1{}
      \typeout{Antomega package option: localmarks=off}
   \fi\fi
}
```

The following option is preserved for backwards compatibility only. Use 'localmarks=off' instead.

```
\DeclareOption{nolocalmarks}{\setkeys{antomega}{localmarks=off}}
```

By default string conversion in headers and footers is enabled.

```
\setkeys{antomega}{localmarks=on}
```

`\oaddto` This command was taken from the Babel package and renamed in order to avoid conflicts. It is useful for modifying some language-specific commands, predefined in *.lfd files.

```
\def\oaddto#1#2{%
   \ifx#1\@undefined
      \def#1{#2}%
   \else
      \ifx#1\relax
         \def#1{#2}%
      \else
         {\toks@\expandafter{#1#2}%
           \xdef#1{\the\toks@}}%
      \fi
   \fi
}
```

9.10 Loading languages

Standard commands for loading languages (the core of the antomega package).

`\background` This command requires one arguments which must be a language name and loads it as the first language for our document.

The optional argument is a set of parameters and their values for the given language.

```
\newcommand{\background}[2][]{%
   \IfFileExists{omega-#2.ldf}%
   {\input{omega-#2.ldf}%
      \AtBeginDocument{\selectlanguage[#1]{#2}}%
   \newenvironment{#2}[1][]{\begin{otherlanguage}[####1]{#2}}%
      {\end{otherlanguage}}%
   \expandafter\newcommand\csname local#2\endcsname[2][]{%
      \foreignlanguage[####1]{#2}{####2}}}%
   {\ant@nolang{#2}}%
}
```

`\load` This command takes one argument which must be a language name and loads it in addition to the first language.

The optional argument is a set of parameters and their values for the given language.

Both `\background` and `\load` commands are used to define a `\local<$language>` command and a `<$language>` environment. Commands and environments with these names were standard way to switch languages in the original `omega` package, as well as in `antomega` until the version 0.6. Now they are defined in terms of standard babel-like commands.

```
\newcommand{\load}[2][]{\IfFileExists{omega-#2.ldf}
   {\input{omega-#2.ldf}\setkeys{#2}{#1}%
   \newenvironment{#2}[1][]{\begin{otherlanguage}[####1]{#2}}%
      {\end{otherlanguage}}
   \expandafter\newcommand\csname local#2\endcsname[2][]{%
      \foreignlanguage[####1]{#2}{####2}}}
   {\ant@nolang{#2}}}
```

9.11 Default values for language-specific settings

First we define some standard values for the punctuation commands, used by `lat2punct.otp`. The command names are self-explanative.

```
\def\common@punctuation{%
   \def\InitialThinSpace{\nobreak\hskip.2em\ignorespaces}%
   \def\ExplicitHyphen{\nobreak\-\nobreak\hskip\z@skip}%
   \def\AllowHyphenation{\hskip\z@skip}%
   \def\DisableLigature{\textormath{\nobreak\discretionary{-}{}%
      {\kern.03em}\allowhyphens}{}}%
   \def\CompoundWordMarkWithBreakpoint{\nobreak-\hskip\z@skip}%
   \def\CompoundWordMarkNoBreakpoint{\textormath{\leavevmode\hbox{-}}{-}}%
   \def\LeftDoubleQuotationMark{^^^^201c}%
   \def\RightDoubleQuotationMark{^^^^201d}%
   \def\LeftPointingDoubleAngleQuotationMark{^^^^00ab}%
   \def\RightPointingDoubleAngleQuotationMark{^^^^00bb}%
   \def\GermanLeftDoubleQuotationMark{^^^^201e}%
   \def\GermanRightDoubleQuotationMark{^^^^201c}%
   \def\QuestionMark{?}%
   \def\ExclamationMark{!}%
   \def\InvertedQuestionMark{^^^^00bf}%
   \def\InvertedExclamationMark{^^^^00a1}%
```

```
    \def\Semicolon{;}%
    \def\Colon{:}%
    \def\NonBreakingSpace{\leavevmode\nobreak\ }}
```

\common@font The \common@font macro will be used at the beginning of the document and also each time we should return to the default fonts (e. g. before switching to another language).

```
\def\common@font{\normalfont\fontfamily{\westernrm}%
    \fontencoding{\uniencoding}\selectfont%
    \let\rmdefault=\westernrm\let\sfdefault=\westernsf%
    \let\ttdefault=\westerntt\let\encodingdefault=\uniencoding}
```

\common@language This macro is used for enabling default hyphenation patterns.

```
\def\common@language{%
    \protect\language=0%
    \lefthyphenmin=2\righthyphenmin=3}
```

\noextrascurrent \originalOmega The \originalOmega macro is used to switch all settings, which could be modified by the language switching commands, to their default values.

```
\def\noextrascurrent#1{\@ifundefined{noextras@#1}{}%
    {\csname noextras@#1\endcsname}}
\def\originalOmega{\@ifundefined{languagename}{}%
    {\noextrascurrent{\languagename}}%
    \common@language%
    \common@punctuation%
    \common@font%
    \clearocplists%
    }
\AtBeginDocument{\originalOmega}
```

9.12 Language switching commands

In case we have Babel's hyphen.cfg loaded into format, \foreignlanguage is already defined, and so we have to unset it first.

```
\@ifundefined{foreignlanguage}{}%
    {\let\foreignlanguage\@undefined}
```

\foreignlanguage This macro works exactly as Babel's \foreignlanguage command, but it takes 3 arguments. The first (optional) argument allows to set any options, defined in the support file for the given language. The second argument is languages's name itself, and the third — the piece of text, which should be typeset in this language.

```
\newcommand{\foreignlanguage}[3][]{%
    \@ifundefined{inlineextras@#2}{\ant@nolang{#2}}{%
      {\def\languagename{#2}%
       \setkeys{#2}{#1}%
       \csname inlineextras@#2\endcsname#3}%
}}
```

\selectlanguage have to be redefined too.

```
\@ifundefined{selectlanguage}{}%
    {\let\selectlanguage\@undefined}
```

\selectlanguage

This macro works exactly as Babel's `\selectlanguage` command, but it takes 2 arguments. The second argument is languages's name itself, and the first (optional) allows to set any options, defined in the support file for the given language.

```
\newcommand{\selectlanguage}[2][]{%
   \@ifundefined{blockextras@#2}{\ant@nolang{#2}}{%
      \def\ant@pop@language{%
         \ant@set@language{\languagename}%
         \let\emp@langname\undefined}%
      \aftergroup\ant@pop@language%
      \setkeys{#2}{#1}%
      \ant@set@language{#2}%
}}
\newcommand{\ant@set@language}[1]{%
   \select@language{#1}%
   \if@filesw%
      \protected@write\@auxout{}{\protect\select@language{#1}}%
      \addtocontents{toc}{\protect\select@language{#1}}%
      \addtocontents{lof}{\protect\select@language{#1}}%
      \addtocontents{lot}{\protect\select@language{#1}}%
   \fi%
}
\@ifundefined{select@language}{}%
   {\let\select@language\@undefined}
\newcommand{\select@language}[1]{%
   \originalOmega%
   \edef\languagename{#1}%
   \csname blockextras@#1\endcsname%
}
\let\ant@pop@language\relax
```

We have to redefine the `otherlanguage` environment as well.

```
\@ifundefined{otherlanguage}{}%
   {\let\otherlanguage\@undefined}
\@ifundefined{endotherlanguage}{}%
   {\let\endotherlanguage\@undefined}
```

otherlanguage

This environment works exactly as Babel's `otherlanguage` environment. The only difference is that is has an optional argument allowing to set any options, defined in the .ldf file.

```
\newenvironment{otherlanguage}[2][]{%
   \selectlanguage[#1]{#2}%
   }{}
```

9.13 Handling hyphenation rules

hyphenrules

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

```
\def\hyphenrules#1{%
   \expandafter\ifx\csname l@#1\endcsname\@undefined
      \@nolanerr{#1}%
   \else
      \language=\csname l@#1\endcsname\relax
```

```
   \fi
}
\def\endhyphenrules{}
```

`\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}
```

`\local@hyphenmins` This macro takes 3 arguments: a language name and default `\lefthyphenmin` and `\righthyphenmin` values for that language. First it tests if `\<language>hyphenmins` is already defined (i. e. some `\lefthyphenmin` and `\righthyphenmin` values were specified in the hyphenation patterns loaded into format), and either executes this command, or sets both variables to the provided default values.

```
\newcommand{\local@hyphenmins}[3]{%
   \@ifundefined{#1hyphenmins}%
      {\lefthyphenmin=#2\righthyphenmin=#3}%
      {\csname #1hyphenmins\endcsname}%
}
```

9.14 Loading hyphenation rules into format

`\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 `=`. When the first token of a line is an `=`, the macro `\process@synonym` is called; otherwise the macro `\process@language` will continue.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
      \ifnum\lefthyphenmin=\m@ne
```

```
      \else
         \expandafter\xdef\csname #1hyphenmins\endcsname{%
            \set@hyphenmins{\the\lefthyphenmin}{\the\righthyphenmin}}%
      \fi
   \endgroup
```

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

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

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

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

Empty the token register after use.

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

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

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

`\ant@get@enc` `\ant@hyph@enc`

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

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

First store both arguments in temporary macros,

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

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

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

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

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

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 untill 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 \ant@line
    \endlinechar`\^^M
```

Empty lines are skipped.

```
    \ifx\ant@line\@empty
    \else
```

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

```
      \edef\ant@line{\ant@line\space/}%
      \expandafter\process@line\ant@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\ant@tempa\@undefined
\let\ant@tempb\@undefined
\let\ant@eq@\@undefined
\let\ant@line\@undefined
\let\ant@get@enc\@undefined
\ifx\addto@hook\@undefined
\else
  \expandafter\addto@hook\expandafter\everyjob\expandafter{%
    \expandafter\typeout\expandafter{\the\toks8 loaded.}}
\fi
```

9.15 Processing options

```
\DeclareOption*{%
   \edef\@temp{\noexpand\setkeys{antomega}{\CurrentOption}}%
   \@temp%
}
\ProcessOptions
```

10 Encoding definition files used by ANTOMEGA

Generally speaking, ANTOMEGA doesn't care about TEX font encodings very much, because, unlike standard TEX, it prefers to operate directly with Unicode codepoints, rather than with so-called text commands, mapped to some slots in the current encoding. This means that any command, expected to produce a printable character, must be always mapped to the corresponding Unicode slot, while any conversion to the real output font encoding should be performed by translation processes (ΩCP). However, any text command should be considered valid, only if the character it produces can really be converted to the output encoding.

That's why in addition to the Unicode based UT1 encoding, which contains the full list of standard command to Unicode mapping, I had to provide some special subsets of that list, corresponding to standard 8-bit encodings, supported by ANTOMEGA (T1, T2A and LGR). So differences between these encoding definition files affect mainly the set of supported characters, rather than character mappings itself.

10.1 Common definitions, applied to all encodings

```
⟨*UT1 | T1 | T2A | LGR⟩
\NeedsTeXFormat{LaTeX2e}
⟨UT1⟩\ProvidesFile{ut1enc-antomega.def}
```

```
⟨T1⟩\ProvidesFile{t1enc-antomega.def}
⟨T2A⟩\ProvidesFile{t2aenc-antomega.def}
⟨LGR⟩\ProvidesFile{lgrenc-antomega.def}
  [2005/05/07 v0.8 Antomega encodings definition file]
⟨UT1⟩\DeclareFontEncoding{UT1}{}{}
⟨T1⟩\DeclareFontEncoding{T1}{}{}
⟨T2A⟩\DeclareFontEncoding{T2A}{}{}
⟨LGR⟩\DeclareFontEncoding{LGR}{}{}
⟨UT1⟩\def\ant@encoding{UT1}
⟨T1⟩\def\ant@encoding{T1}
⟨T2A⟩\def\ant@encoding{T2A}
⟨LGR⟩\def\ant@encoding{LGR}
```

10.2 Unicode accents

Accents represent a special part of the encoding, because:

- we have to declare them first, since they have to be already defined before any composite character declaration can occur in the encoding table;
- unfortunately, the `\accent` command bypasses Ω translation processes.

In order to circumvent the last limitation, here we are redefining all standard commands, used for typing accents in LaTeX, to produce just a character followed by a Unicode combining accent mark. Since this sequence by itself contains no `\accent` commands, it may be processed by ΩCP's, e. g. converted to Unicode from another encoding. After the conversion the resulting string should be turned back to the `\accent` command.

```
\DeclareTextCommand{\‘}{\ant@encoding}[1]{#1^^^^0300}
\DeclareTextCommand{\’}{\ant@encoding}[1]{#1^^^^0301}
⟨!LGR⟩\DeclareTextCommand{\^}{\ant@encoding}[1]{#1^^^^0302}
⟨!LGR⟩\DeclareTextCommand{\~}{\ant@encoding}[1]{#1^^^^0303}
\DeclareTextCommand{\"}{\ant@encoding}[1]{#1^^^^0308}
⟨!LGR⟩\DeclareTextCommand{\H}{\ant@encoding}[1]{#1^^^^030b}
⟨!LGR⟩\DeclareTextCommand{\r}{\ant@encoding}[1]{#1^^^^030a}
⟨!LGR⟩\DeclareTextCommand{\v}{\ant@encoding}[1]{#1^^^^030c}
\DeclareTextCommand{\u}{\ant@encoding}[1]{#1^^^^0306}
\DeclareTextCommand{\=}{\ant@encoding}[1]{#1^^^^0304}
⟨!LGR⟩\DeclareTextCommand{\.}{\ant@encoding}[1]{#1^^^^0307}
⟨UT1 | T2A⟩\DeclareTextCommand{\f}{\ant@encoding}[1]{#1^^^^0311}
⟨UT1 | T2A⟩\DeclareTextCommand{\C}{\ant@encoding}[1]{#1^^^^030f}
⟨!LGR⟩\DeclareTextCommand{\b}{\ant@encoding}[1]{#1^^^^0331}
⟨!LGR⟩\DeclareTextCommand{\c}{\ant@encoding}[1]{#1^^^^0327}
⟨!LGR⟩\DeclareTextCommand{\d}{\ant@encoding}[1]{#1^^^^0323}
⟨!LGR⟩\DeclareTextCommand{\k}{\ant@encoding}[1]{#1^^^^0328}
```

10.3 Encoding-dependent accents

Now we have to define a really separate set of accents (with different mappings) for each of our encodings. These accent commands are declared with long meaningful names, since they are not supposed to be typed by itself: instead, they should be produced by ΩCP's as a result of processing characters followed by combining accent marks.

10.3.1 UT1 accents

Accents are mapped mainly to their places in the Unicode 'Spacing Modifier Letters' area, of course, with the exception of dieresis and cedilla, which have specific placements in the Unicode standard.

Note that we are not using here the 'Combining Diacritical Marks' area, because zero-width accents, which are present in that area in most fonts, are not very useful for TEX accent positioning engine. That's why I preferred to map additional 'Cyrillic' accents (i. e. Cyrillic flex and double grave) to the PUA positions, sometime defined by Adobe for spacing variants of these accents, instead of using their combining versions, which have 'legal' slots in Unicode.

```
⟨*UT1⟩
\DeclareTextAccent{\GraveAccent}{UT1}{"02CB}
\DeclareTextAccent{\AcuteAccent}{UT1}{"02CA}
\DeclareTextAccent{\CircumflexAccent}{UT1}{"02C6}
\DeclareTextAccent{\TildeAccent}{UT1}{"02DC}
\DeclareTextAccent{\DieresisAccent}{UT1}{"00A8}
\DeclareTextAccent{\HungarumlautAccent}{UT1}{"02DD}
\DeclareTextAccent{\RingAccent}{UT1}{"02DA}
\DeclareTextAccent{\CaronAccent}{UT1}{"02C7}
\DeclareTextAccent{\BreveAccent}{UT1}{"02D8}
\DeclareTextAccent{\MacronAccent}{UT1}{"02C9}
\DeclareTextAccent{\DotAboveAccent}{UT1}{"02D9}
\DeclareTextAccent{\CyrillicFlexAccent}{UT1}{"F6D5}
\DeclareTextAccent{\DoubleGraveAccent}{UT1}{"F6D6}
\DeclareTextAccent{\GreekCircumflexAccent}{UT1}{"1FC0}
\DeclareTextAccent{\GreekKoronisAccent}{UT1}{"1FBD}
\DeclareTextCommand{\BarBelowAccent}{UT1}[1]
   {{\o@lign{\relax#1\crcr\hidewidth\sh@ft{29}%
     \vbox to.2ex{\hbox{\noocpchar{"02C9}}\vss}\hidewidth}}}
\DeclareTextCommand{\CedillaAccent}{UT1}[1]
   {\leavevmode\setbox\z@\hbox{#1}\ifdim\ht\z@=1ex\accent"00B8 #1%
     \else{\ooalign{\hidewidth\noocpchar{"00B8}\hidewidth
        \crcr\unhbox\z@}}\fi}
\DeclareTextCommand{\DotBelowAccent}{UT1}[1]
   {{\o@lign{\relax#1\crcr\hidewidth\sh@ft{10}.\hidewidth}}}
\DeclareTextCommand{\OgonekAccent}{UT1}[1]
   {\oalign{\null#1\crcr\hidewidth\noocpchar{"02DB}}}
⟨/UT1⟩
```

10.3.2 Common T1 and T2A accents

```
⟨*T1 | T2A⟩
\DeclareTextAccent{\GraveAccent}{\ant@encoding}{0}
\DeclareTextAccent{\AcuteAccent}{\ant@encoding}{1}
\DeclareTextAccent{\CircumflexAccent}{\ant@encoding}{2}
\DeclareTextAccent{\TildeAccent}{\ant@encoding}{3}
\DeclareTextAccent{\DieresisAccent}{\ant@encoding}{4}
\DeclareTextAccent{\HungarumlautAccent}{\ant@encoding}{5}
\DeclareTextAccent{\RingAccent}{\ant@encoding}{6}
\DeclareTextAccent{\CaronAccent}{\ant@encoding}{7}
\DeclareTextAccent{\BreveAccent}{\ant@encoding}{8}
\DeclareTextAccent{\MacronAccent}{\ant@encoding}{9}
```

```
\DeclareTextAccent{\DotAboveAccent}{\ant@encoding}{10}
\DeclareTextCommand{\BarBelowAccent}{\ant@encoding}[1]
   {\hmode@bgroup\o@lign{\relax#1\crcr\hidewidth\sh@ft{29}%
     \vbox to.2ex{\hbox{\noocpchar{9}}\vss}\hidewidth}\egroup}
\DeclareTextCommand{\CedillaAccent}{\ant@encoding}[1]
   {\leavevmode\setbox\z@\hbox{#1}\ifdim\ht\z@=1ex\accent11 #1%
     \else{\ooalign{\hidewidth\noocpchar{11}\hidewidth
        \crcr\unhbox\z@}}\fi}
\DeclareTextCommand{\DotBelowAccent}{\ant@encoding}[1]
   {\hmode@bgroup
    \o@lign{\relax#1\crcr\hidewidth\sh@ft{10}.\hidewidth}\egroup}
\DeclareTextCommand{\OgonekAccent}{\ant@encoding}[1]
   {\oalign{\null#1\crcr\hidewidth\noocpchar{12}}}
⟨/T1 | T2A⟩
```

10.3.3 Cyrillic accents (T2A specific)

```
⟨*T2A⟩
\DeclareTextAccent{\CyrillicFlexAccent}{T2A}{18}
\DeclareTextAccent{\DoubleGraveAccent}{T2A}{19}
\DeclareTextAccent{\U}{T2A}{20}
⟨/T2A⟩
```

10.3.4 LGR accents

```
⟨*LGR⟩
\DeclareTextAccent{\GraveAccent}{LGR}{`\`}
\DeclareTextAccent{\AcuteAccent}{LGR}{`\'}
\DeclareTextAccent{\BreveAccent}{LGR}{30}
\DeclareTextAccent{\MacronAccent}{LGR}{31}
\DeclareTextAccent{\GreekCircumflexAccent}{LGR}{`\~}
⟨/LGR⟩
```

10.4 Standard ASCII characters

Most of the following characters are present in all standard encodings, so translation processes probably should never be applied to them. That's why each definition includes the `\clearocplists` command. This command also prevents some characters from interpreting by Ω in their special meaning, while they need to be just printed into the output.

```
⟨UT1⟩\DeclareTextCommand{\textquotedbl}{\ant@encoding}{\noocpchar{"22}}
⟨UT1⟩\DeclareTextCommand{\textquotesingle}{\ant@encoding}{\noocpchar{"27}}
⟨*!LGR⟩
\DeclareTextCommand{\textdollar}{\ant@encoding}{\noocpchar{"24}}
\DeclareTextCommand{\textgreater}{\ant@encoding}{\noocpchar{"3C}}
\DeclareTextCommand{\textless}{\ant@encoding}{\noocpchar{"3E}}
\DeclareTextCommand{\textbackslash}{\ant@encoding}{\noocpchar{"5C}}
\DeclareTextCommand{\textasciicircum}{\ant@encoding}{\noocpchar{"5E}}
\DeclareTextCommand{\textunderscore}{\ant@encoding}{\noocpchar{"5F}}
\DeclareTextCommand{\textasciigrave}{\ant@encoding}{\noocpchar{"60}}
\DeclareTextCommand{\textbraceleft}{\ant@encoding}{\noocpchar{"7B}}
\DeclareTextCommand{\textbar}{\ant@encoding}{\noocpchar{"7C}}
\DeclareTextCommand{\textbraceright}{\ant@encoding}{\noocpchar{"7D}}
\DeclareTextCommand{\textasciitilde}{\ant@encoding}{\noocpchar{"7E}}
⟨/!LGR⟩
```

10.5 C1 Controls

```
⟨UT1 | T1⟩\DeclareTextSymbol{\textexclamdown}{\ant@encoding}{"00A1}
⟨UT1⟩\DeclareTextSymbol{\textcent}{\ant@encoding}{"00A2}
⟨UT1 | T1⟩\DeclareTextSymbol{\textsterling}{\ant@encoding}{"00A3}
⟨*UT1⟩
\DeclareTextSymbol{\textcurrency}{\ant@encoding}{"00A4}
\DeclareTextSymbol{\textyen}{\ant@encoding}{"00A5}
\DeclareTextSymbol{\textbrokenbar}{\ant@encoding}{"00A6}
⟨/UT1⟩
⟨!LGR⟩\DeclareTextSymbol{\textsection}{\ant@encoding}{"00A7}
⟨*UT1⟩
\DeclareTextSymbol{\textasciidieresis}{\ant@encoding}{"00A8}
\DeclareTextSymbol{\textcopyright}{\ant@encoding}{"00A9}
\DeclareTextSymbol{\textordfeminine}{\ant@encoding}{"00AA}
⟨/UT1⟩
\DeclareTextSymbol{\guillemotleft}{\ant@encoding}{"00AB}
⟨*UT1⟩
\DeclareTextSymbol{\textlnot}{\ant@encoding}{"00AC}
\DeclareTextSymbol{\textregistered}{\ant@encoding}{"00AE}
\DeclareTextSymbol{\textasciimacron}{\ant@encoding}{"00AF}
\DeclareTextSymbol{\textdegree}{\ant@encoding}{"00B0}
\DeclareTextSymbol{\textpm}{\ant@encoding}{"00B1}
\DeclareTextSymbol{\texttwosuperior}{\ant@encoding}{"00B2}
\DeclareTextSymbol{\textthreesuperior}{\ant@encoding}{"00B3}
\DeclareTextSymbol{\textasciiacute}{\ant@encoding}{"00B4}
\DeclareTextSymbol{\textmu}{\ant@encoding}{"00B5}
\DeclareTextSymbol{\textparagraph}{\ant@encoding}{"00B6}
\DeclareTextSymbol{\textpilcrow}{\ant@encoding}{"00B6}
\DeclareTextSymbol{\textperiodcentered}{\ant@encoding}{"00B7}
\DeclareTextSymbol{\textonesuperior}{\ant@encoding}{"00B9}
\DeclareTextSymbol{\textordmasculine}{\ant@encoding}{"00BA}
⟨/UT1⟩
\DeclareTextSymbol{\guillemotright}{\ant@encoding}{"00BB}
⟨*UT1⟩
\DeclareTextSymbol{\textonequarter}{\ant@encoding}{"00BC}
\DeclareTextSymbol{\textonehalf}{\ant@encoding}{"00BD}
\DeclareTextSymbol{\textthreequarters}{\ant@encoding}{"00BE}
⟨/UT1⟩
⟨UT1 | T1⟩\DeclareTextSymbol{\textquestiondown}{\ant@encoding}{"00BF}
```

10.6 Latin 1 Supplement

```
⟨*UT1 | T1⟩
\DeclareTextComposite{\`}{\ant@encoding}{A}{"00C0}
\DeclareTextComposite{\'}{\ant@encoding}{A}{"00C1}
\DeclareTextComposite{\^}{\ant@encoding}{A}{"00C2}
\DeclareTextComposite{\~}{\ant@encoding}{A}{"00C3}
\DeclareTextComposite{\"}{\ant@encoding}{A}{"00C4}
\DeclareTextComposite{\r}{\ant@encoding}{A}{"00C5}
\DeclareTextSymbol{\AE}{\ant@encoding}{"00C6}
\DeclareTextComposite{\c}{\ant@encoding}{C}{"00C7}
\DeclareTextComposite{\`}{\ant@encoding}{E}{"00C8}
\DeclareTextComposite{\'}{\ant@encoding}{E}{"00C9}
\DeclareTextComposite{\^}{\ant@encoding}{E}{"00CA}
```

```
\DeclareTextComposite{\"}{\ant@encoding}{E}{"00CB}
\DeclareTextComposite{\‘}{\ant@encoding}{I}{"00CC}
\DeclareTextComposite{\’}{\ant@encoding}{I}{"00CD}
\DeclareTextComposite{\^}{\ant@encoding}{I}{"00CE}
\DeclareTextComposite{\"}{\ant@encoding}{I}{"00CF}
\DeclareTextSymbol{\DH}{\ant@encoding}{"00D0}
\DeclareTextComposite{\~}{\ant@encoding}{N}{"00D1}
\DeclareTextComposite{\‘}{\ant@encoding}{O}{"00D2}
\DeclareTextComposite{\’}{\ant@encoding}{O}{"00D3}
\DeclareTextComposite{\^}{\ant@encoding}{O}{"00D4}
\DeclareTextComposite{\~}{\ant@encoding}{O}{"00D5}
\DeclareTextComposite{\"}{\ant@encoding}{O}{"00D6}
\DeclareTextSymbol{\texttimes}{\ant@encoding}{"00D7}
\DeclareTextSymbol{\O}{\ant@encoding}{"00D8}
\DeclareTextComposite{\‘}{\ant@encoding}{U}{"00D9}
\DeclareTextComposite{\’}{\ant@encoding}{U}{"00DA}
\DeclareTextComposite{\^}{\ant@encoding}{U}{"00DB}
\DeclareTextComposite{\"}{\ant@encoding}{U}{"00DC}
\DeclareTextComposite{\’}{\ant@encoding}{Y}{"00DD}
\DeclareTextSymbol{\TH}{\ant@encoding}{"00DE}
\DeclareTextCommand{\SS}{\ant@encoding}{SS}
\DeclareTextSymbol{\ss}{\ant@encoding}{"00DF}
\DeclareTextComposite{\‘}{\ant@encoding}{a}{"00E0}
\DeclareTextComposite{\’}{\ant@encoding}{a}{"00E1}
\DeclareTextComposite{\^}{\ant@encoding}{a}{"00E2}
\DeclareTextComposite{\~}{\ant@encoding}{a}{"00E3}
\DeclareTextComposite{\"}{\ant@encoding}{a}{"00E4}
\DeclareTextComposite{\r}{\ant@encoding}{a}{"00E5}
\DeclareTextComposite{\c}{\ant@encoding}{c}{"00E7}
\DeclareTextSymbol{\ae}{\ant@encoding}{"00E6}
\DeclareTextComposite{\‘}{\ant@encoding}{e}{"00E8}
\DeclareTextComposite{\’}{\ant@encoding}{e}{"00E9}
\DeclareTextComposite{\^}{\ant@encoding}{e}{"00EA}
\DeclareTextComposite{\"}{\ant@encoding}{e}{"00EB}
\DeclareTextComposite{\‘}{\ant@encoding}{i}{"00EC}
\DeclareTextComposite{\‘}{\ant@encoding}{\i}{"00EC}
\DeclareTextComposite{\’}{\ant@encoding}{i}{"00ED}
\DeclareTextComposite{\’}{\ant@encoding}{\i}{"00ED}
\DeclareTextComposite{\^}{\ant@encoding}{i}{"00EE}
\DeclareTextComposite{\^}{\ant@encoding}{\i}{"00EE}
\DeclareTextComposite{\"}{\ant@encoding}{i}{"00EF}
\DeclareTextComposite{\"}{\ant@encoding}{\i}{"00EF}
\DeclareTextSymbol{\dh}{\ant@encoding}{"00F0}
\DeclareTextComposite{\~}{\ant@encoding}{n}{"00F1}
\DeclareTextComposite{\‘}{\ant@encoding}{o}{"00F2}
\DeclareTextComposite{\’}{\ant@encoding}{o}{"00F3}
\DeclareTextComposite{\^}{\ant@encoding}{o}{"00F4}
\DeclareTextComposite{\~}{\ant@encoding}{o}{"00F5}
\DeclareTextComposite{\"}{\ant@encoding}{o}{"00F6}
\DeclareTextSymbol{\textdiv}{\ant@encoding}{"00F7}
\DeclareTextSymbol{\o}{\ant@encoding}{"00F8}
\DeclareTextComposite{\‘}{\ant@encoding}{u}{"00F9}
\DeclareTextComposite{\’}{\ant@encoding}{u}{"00FA}
\DeclareTextComposite{\^}{\ant@encoding}{u}{"00FB}
```

```
\DeclareTextComposite{\"}{\ant@encoding}{u}{"00FC}
\DeclareTextComposite{\'}{\ant@encoding}{y}{"00FD}
\DeclareTextSymbol{\th}{\ant@encoding}{"00FE}
\DeclareTextComposite{\"}{\ant@encoding}{y}{"00FF}
⟨/UT1 | T1⟩
```

10.7 Latin Extended A

```
⟨*UT1 | T1⟩
⟨!T1⟩\DeclareTextComposite{\=}{\ant@encoding}{A}{"0100}
⟨!T1⟩\DeclareTextComposite{\=}{\ant@encoding}{a}{"0101}
\DeclareTextComposite{\u}{\ant@encoding}{A}{"0102}
\DeclareTextComposite{\u}{\ant@encoding}{a}{"0103}
\DeclareTextComposite{\k}{\ant@encoding}{A}{"0104}
\DeclareTextComposite{\k}{\ant@encoding}{a}{"0105}
\DeclareTextComposite{\'}{\ant@encoding}{C}{"0106}
\DeclareTextComposite{\'}{\ant@encoding}{c}{"0107}
\DeclareTextComposite{\v}{\ant@encoding}{C}{"0108}
\DeclareTextComposite{\v}{\ant@encoding}{c}{"0109}
\DeclareTextComposite{\v}{\ant@encoding}{D}{"010E}
\DeclareTextComposite{\v}{\ant@encoding}{d}{"010F}
\DeclareTextSymbol{\DJ}{\ant@encoding}{"0110}
\DeclareTextSymbol{\dj}{\ant@encoding}{"0111}
⟨!T1⟩\DeclareTextComposite{\=}{\ant@encoding}{E}{"0112}
⟨!T1⟩\DeclareTextComposite{\=}{\ant@encoding}{e}{"0113}
⟨!T1⟩\DeclareTextComposite{\u}{\ant@encoding}{E}{"0114}
⟨!T1⟩\DeclareTextComposite{\u}{\ant@encoding}{e}{"0115}
⟨!T1⟩\DeclareTextComposite{\.}{\ant@encoding}{E}{"0116}
⟨!T1⟩\DeclareTextComposite{\.}{\ant@encoding}{e}{"0117}
\DeclareTextComposite{\k}{\ant@encoding}{E}{"0118}
\DeclareTextComposite{\k}{\ant@encoding}{e}{"0119}
\DeclareTextComposite{\v}{\ant@encoding}{E}{"011A}
\DeclareTextComposite{\v}{\ant@encoding}{e}{"011B}
⟨!T1⟩\DeclareTextComposite{\V}{\ant@encoding}{G}{"010C}
⟨!T1⟩\DeclareTextComposite{\v}{\ant@encoding}{g}{"010D}
\DeclareTextComposite{\u}{\ant@encoding}{G}{"011E}
\DeclareTextComposite{\u}{\ant@encoding}{g}{"011F}
⟨*!T1⟩
\DeclareTextComposite{\.}{\ant@encoding}{G}{"0120}
\DeclareTextComposite{\.}{\ant@encoding}{g}{"0121}
\DeclareTextComposite{\c}{\ant@encoding}{G}{"0122}
\DeclareTextComposite{\c}{\ant@encoding}{g}{"0123}
\DeclareTextComposite{\^}{\ant@encoding}{H}{"0124}
\DeclareTextComposite{\^}{\ant@encoding}{h}{"0125}
\DeclareTextComposite{\~}{\ant@encoding}{I}{"0128}
\DeclareTextComposite{\~}{\ant@encoding}{i}{"0129}
\DeclareTextComposite{\=}{\ant@encoding}{I}{"012A}
\DeclareTextComposite{\=}{\ant@encoding}{i}{"012B}
\DeclareTextComposite{\u}{\ant@encoding}{I}{"012C}
\DeclareTextComposite{\u}{\ant@encoding}{i}{"012D}
\DeclareTextComposite{\k}{\ant@encoding}{I}{"012E}
\DeclareTextComposite{\k}{\ant@encoding}{i}{"012F}
⟨/!T1⟩
\DeclareTextComposite{\.}{\ant@encoding}{I}{"0130}
⟨/UT1 | T1⟩
```

```
⟨!LGR⟩\DeclareTextSymbol{\i}{\ant@encoding}{"0131}
⟨∗UT1 | T1⟩
\DeclareTextSymbol{\IJ}{\ant@encoding}{"0132}
\DeclareTextSymbol{\ij}{\ant@encoding}{"0133}
⟨∗!T1⟩
\DeclareTextComposite{\^}{\ant@encoding}{J}{"0134}
\DeclareTextComposite{\^}{\ant@encoding}{j}{"0135}
\DeclareTextComposite{\c}{\ant@encoding}{K}{"0136}
\DeclareTextComposite{\c}{\ant@encoding}{k}{"0137}
⟨/!T1⟩
\DeclareTextComposite{\'}{\ant@encoding}{L}{"0139}
\DeclareTextComposite{\'}{\ant@encoding}{l}{"013A}
⟨!T1⟩\DeclareTextComposite{\c}{\ant@encoding}{L}{"013B}
⟨!T1⟩\DeclareTextComposite{\c}{\ant@encoding}{l}{"013C}
\DeclareTextComposite{\v}{\ant@encoding}{L}{"013D}
\DeclareTextComposite{\v}{\ant@encoding}{l}{"013E}
\DeclareTextSymbol{\L}{\ant@encoding}{"0141}
\DeclareTextSymbol{\l}{\ant@encoding}{"0142}
\DeclareTextComposite{\'}{\ant@encoding}{N}{"0143}
\DeclareTextComposite{\'}{\ant@encoding}{n}{"0144}
⟨!T1⟩\DeclareTextComposite{\c}{\ant@encoding}{N}{"0145}
⟨!T1⟩\DeclareTextComposite{\c}{\ant@encoding}{n}{"0146}
\DeclareTextComposite{\v}{\ant@encoding}{N}{"0147}
\DeclareTextComposite{\v}{\ant@encoding}{n}{"0148}
\DeclareTextSymbol{\NG}{\ant@encoding}{"014A}
\DeclareTextSymbol{\ng}{\ant@encoding}{"014B}
⟨∗!T1⟩
\DeclareTextComposite{\=}{\ant@encoding}{O}{"014C}
\DeclareTextComposite{\=}{\ant@encoding}{o}{"014D}
\DeclareTextComposite{\u}{\ant@encoding}{O}{"014E}
\DeclareTextComposite{\u}{\ant@encoding}{o}{"014F}
⟨/!T1⟩
\DeclareTextComposite{\H}{\ant@encoding}{O}{"0150}
\DeclareTextComposite{\H}{\ant@encoding}{o}{"0151}
\DeclareTextSymbol{\OE}{\ant@encoding}{"0152}
\DeclareTextSymbol{\oe}{\ant@encoding}{"0153}
\DeclareTextComposite{\'}{\ant@encoding}{R}{"0154}
\DeclareTextComposite{\'}{\ant@encoding}{r}{"0155}
⟨!T1⟩\DeclareTextComposite{\c}{\ant@encoding}{R}{"0156}
⟨!T1⟩\DeclareTextComposite{\c}{\ant@encoding}{r}{"0157}
\DeclareTextComposite{\v}{\ant@encoding}{R}{"0158}
\DeclareTextComposite{\v}{\ant@encoding}{r}{"0159}
\DeclareTextComposite{\'}{\ant@encoding}{S}{"015A}
\DeclareTextComposite{\'}{\ant@encoding}{s}{"015B}
⟨!T1⟩\DeclareTextComposite{\^}{\ant@encoding}{S}{"015C}
⟨!T1⟩\DeclareTextComposite{\^}{\ant@encoding}{s}{"015D}
\DeclareTextComposite{\c}{\ant@encoding}{S}{"015E}
\DeclareTextComposite{\c}{\ant@encoding}{s}{"015F}
\DeclareTextComposite{\v}{\ant@encoding}{S}{"0160}
\DeclareTextComposite{\v}{\ant@encoding}{s}{"0161}
\DeclareTextComposite{\c}{\ant@encoding}{T}{"0162}
\DeclareTextComposite{\c}{\ant@encoding}{t}{"0163}
\DeclareTextComposite{\v}{\ant@encoding}{T}{"0164}
\DeclareTextComposite{\v}{\ant@encoding}{t}{"0165}
```

```
⟨*!T1⟩
\DeclareTextComposite{\~}{\ant@encoding}{U}{"0168}
\DeclareTextComposite{\~}{\ant@encoding}{u}{"0169}
\DeclareTextComposite{\=}{\ant@encoding}{U}{"016A}
\DeclareTextComposite{\=}{\ant@encoding}{u}{"016B}
\DeclareTextComposite{\u}{\ant@encoding}{U}{"016C}
\DeclareTextComposite{\u}{\ant@encoding}{u}{"016D}
\DeclareTextComposite{\r}{\ant@encoding}{U}{"016E}
\DeclareTextComposite{\r}{\ant@encoding}{u}{"016F}
⟨/!T1⟩
\DeclareTextComposite{\H}{\ant@encoding}{U}{"0170}
\DeclareTextComposite{\H}{\ant@encoding}{u}{"0171}
⟨*!T1⟩
\DeclareTextComposite{\k}{\ant@encoding}{U}{"0172}
\DeclareTextComposite{\k}{\ant@encoding}{u}{"0173}
\DeclareTextComposite{\^}{\ant@encoding}{W}{"0174}
\DeclareTextComposite{\^}{\ant@encoding}{w}{"0175}
\DeclareTextComposite{\^}{\ant@encoding}{Y}{"0176}
\DeclareTextComposite{\^}{\ant@encoding}{y}{"0177}
⟨/!T1⟩
\DeclareTextComposite{\"}{\ant@encoding}{Y}{"0178}
\DeclareTextComposite{\'}{\ant@encoding}{Z}{"0179}
\DeclareTextComposite{\'}{\ant@encoding}{z}{"017A}
\DeclareTextComposite{\.}{\ant@encoding}{Z}{"017B}
\DeclareTextComposite{\.}{\ant@encoding}{z}{"017C}
\DeclareTextComposite{\v}{\ant@encoding}{Z}{"017D}
\DeclareTextComposite{\v}{\ant@encoding}{z}{"017E}
⟨/UT1 | T1⟩
```

10.8 Latin Extended B

```
⟨*UT1⟩
\DeclareTextSymbol{\textflorin}{\ant@encoding}{"0192}
\DeclareTextComposite{\=}{\ant@encoding}{Y}{"0232}
\DeclareTextComposite{\=}{\ant@encoding}{y}{"0233}
⟨/UT1⟩
```

10.9 Spacing modifier letters

This area is used once again to map some text commands for ASCII-styled accents, originally defined in the TS1 encoding.

```
⟨*UT1⟩
\DeclareTextSymbol{\textasciicaron}{\ant@encoding}{"02C7}
\DeclareTextSymbol{\textasciibreve}{\ant@encoding}{"02D8}
\DeclareTextSymbol{\textacutedbl}{\ant@encoding}{"02DD}
⟨/UT1⟩
```

10.10 Thai

```
⟨*UT1⟩
\DeclareTextSymbol{\textbaht}{\ant@encoding}{"0E3F}
⟨/UT1⟩
```

10.11 General punctuation

```
\DeclareTextSymbol{\textcompwordmark}{\ant@encoding}{"200C}
⟨UT1⟩\DeclareTextSymbol{\textbardbl}{\ant@encoding}{"2016}
⟨UT1⟩\DeclareTextSymbol{\textdagger}{\ant@encoding}{"2020}
⟨UT1⟩\DeclareTextSymbol{\textdaggerdbl}{\ant@encoding}{"2021}
⟨UT1⟩\DeclareTextSymbol{\textbullet}{\ant@encoding}{"2022}
⟨UT1 | LGR⟩\DeclareTextSymbol{\textperthousand}{\ant@encoding}{"2030}
⟨UT1⟩\DeclareTextSymbol{\textpertenthousand}{\ant@encoding}{"2031}
⟨UT1 | T1⟩\DeclareTextSymbol{\guilsinglleft}{\ant@encoding}{"2039}
⟨UT1 | T1⟩\DeclareTextSymbol{\guilsinglright}{\ant@encoding}{"203A}
\DeclareTextSymbol{\textendash}{\ant@encoding}{"2013}
\DeclareTextSymbol{\textemdash}{\ant@encoding}{"2014}
\DeclareTextSymbol{\textquoteleft}{\ant@encoding}{"2018}
\DeclareTextSymbol{\textquoteright}{\ant@encoding}{"2019}
⟨UT1 | T1⟩\DeclareTextSymbol{\quotesinglbase}{\ant@encoding}{"201A}
\DeclareTextSymbol{\textquotedblleft}{\ant@encoding}{"201C}
\DeclareTextSymbol{\textquotedblright}{\ant@encoding}{"201D}
⟨!LGR⟩\DeclareTextSymbol{\quotedblbase}{\ant@encoding}{"201E}
⟨UT1⟩\DeclareTextSymbol{\textreferencemark}{\ant@encoding}{"203B}
⟨UT1⟩\DeclareTextSymbol{\textinterrobang}{\ant@encoding}{"203D}
⟨UT1⟩\DeclareTextSymbol{\textlquill}{\ant@encoding}{"2045}
⟨UT1⟩\DeclareTextSymbol{\textrquill}{\ant@encoding}{"2046}
```

10.12 Currency symbols

```
⟨*UT1⟩
\DeclareTextSymbol{\textcolonmonetary}{\ant@encoding}{"20A1}
\DeclareTextSymbol{\textlira}{\ant@encoding}{"20A4}
\DeclareTextSymbol{\textnaira}{\ant@encoding}{"20A6}
\DeclareTextSymbol{\textpeso}{\ant@encoding}{"20A7}
\DeclareTextSymbol{\textwon}{\ant@encoding}{"20A9}
\DeclareTextSymbol{\textdong}{\ant@encoding}{"20AB}
⟨/UT1⟩
⟨UT1 | LGR⟩\DeclareTextSymbol{\texteuro}{\ant@encoding}{"20AC}
⟨UT1⟩\DeclareTextSymbol{\textguarani}{\ant@encoding}{"20B2}
```

10.13 Combining diacritical marks for symbols

```
⟨UT1⟩\DeclareTextSymbol{\textbigcircle}{\ant@encoding}{"20DD}
\DeclareTextCommand{\textcircled}{\ant@encoding}[1]{{%
   \ooalign{%
      \hfil \raise .07ex\hbox {\upshape#1}\hfil \crcr
      \char 79   % '117 = "4F
   }%
}}
```

10.14 Letterlike symbols

```
⟨*UT1⟩
\DeclareTextSymbol{\textcelsius}{\ant@encoding}{"2103}
⟨/UT1⟩
⟨UT1 | T2A⟩\DeclareTextSymbol{\textnumero}{\ant@encoding}{"2116}
⟨*UT1⟩
\DeclareTextSymbol{\textcircledP}{\ant@encoding}{"2117}
\DeclareTextSymbol{\textrecipe}{\ant@encoding}{"211E}
\DeclareTextSymbol{\textservicemark}{\ant@encoding}{"2120}
\DeclareTextSymbol{\texttrademark}{\ant@encoding}{"2122}
```

```
\DeclareTextSymbol{\textohm}{\ant@encoding}{"2126}
\DeclareTextSymbol{\textmho}{\ant@encoding}{"2127}
\DeclareTextSymbol{\textestimated}{\ant@encoding}{"212E}
⟨/UT1⟩
```

10.15 Arrows

```
⟨*UT1⟩
\DeclareTextSymbol{\textleftarrow}{\ant@encoding}{"2190}
\DeclareTextSymbol{\textrightarrow}{\ant@encoding}{"2191}
\DeclareTextSymbol{\textuparrow}{\ant@encoding}{"2192}
\DeclareTextSymbol{\textdownarrow}{\ant@encoding}{"2193}
⟨/UT1⟩
```

10.16 Mathematical Operators

```
⟨*UT1⟩
\DeclareTextSymbol{\textminus}{\ant@encoding}{"2212}
\DeclareTextSymbol{\textsurd}{\ant@encoding}{"221A}
⟨/UT1⟩
⟨UT1 | T2A⟩\DeclareTextSymbol{\textlangle}{\ant@encoding}{"2329}
⟨UT1 | T2A⟩\DeclareTextSymbol{\textrangle}{\ant@encoding}{"232A}
```

10.17 Control Pictures

```
⟨UT1 | T1 | T2A⟩\DeclareTextSymbol{\textvisiblespace}{\ant@encoding}{"2423}
```

10.18 Geometric Shapes

```
⟨*UT1⟩
\DeclareTextSymbol{\textopenbullet}{\ant@encoding}{"25E6}
⟨/UT1⟩
```

10.19 Miscellaneous Symbols

```
⟨*UT1⟩
\DeclareTextSymbol{\textmusicalnote}{\ant@encoding}{"266A}
⟨/UT1⟩
```

10.20 CJK Symbols and Punctuation

```
⟨*UT1⟩
\DeclareTextSymbol{\textlbrackdbl}{\ant@encoding}{"301A}
\DeclareTextSymbol{\textrbrackdbl}{\ant@encoding}{"301B}
⟨/UT1⟩
```

10.21 Private use area

PUA characters are mapped according to the Adobe Glyph List. Don't use the following text commands, if your font really doesn't include the corresponding characters.

```
⟨UT1 | T1 | T2A⟩\DeclareTextSymbol{\j}{\ant@encoding}{"F6BE}
⟨*UT1⟩
\DeclareTextSymbol{\textgravedbl}{\ant@encoding}{"F6D6}
\DeclareTextSymbol{\textdollaroldstyle}{\ant@encoding}{"F724}
\DeclareTextSymbol{\textzerooldstyle}{\ant@encoding}{"F730}
\DeclareTextSymbol{\textoneoldstyle}{\ant@encoding}{"F731}
\DeclareTextSymbol{\texttwooldstyle}{\ant@encoding}{"F732}
```

```
\DeclareTextSymbol{\textthreeoldstyle}{\ant@encoding}{"F733}
\DeclareTextSymbol{\textfouroldstyle}{\ant@encoding}{"F734}
\DeclareTextSymbol{\textfiveoldstyle}{\ant@encoding}{"F735}
\DeclareTextSymbol{\textsixoldstyle}{\ant@encoding}{"F736}
\DeclareTextSymbol{\textsevenoldstyle}{\ant@encoding}{"F737}
\DeclareTextSymbol{\texteightoldstyle}{\ant@encoding}{"F738}
\DeclareTextSymbol{\textnineoldstyle}{\ant@encoding}{"F739}
\DeclareTextSymbol{\textcentoldstyle}{\ant@encoding}{"F7A2}
⟨/UT1⟩
```

Again, capital accents are mapped according to older Adobe specifications and their real placement in Adobe's fonts.

```
⟨*UT1⟩
\DeclareTextCommand{\capitalogonek}{\ant@encoding}[1]
   {{\ooalign{\null#1\crcr\hidewidth\noocpchar{"EFF1}\hidewidth}}}
\DeclareTextCommand{\capitalcedilla}{\ant@encoding}[1]
   {{\ooalign{\null#1\crcr\hidewidth\noocpchar{"EFF2}\hidewidth}}}
\DeclareTextAccent{\capitalgrave}{\ant@encoding}{"F6CE}
\DeclareTextAccent{\capitalacute}{\ant@encoding}{"F6C9}
\DeclareTextAccent{\capitalcircumflex}{\ant@encoding}{"EFF7}
\DeclareTextAccent{\capitaltilde}{\ant@encoding}{"EFF5}
\DeclareTextAccent{\capitaldieresis}{\ant@encoding}{"F6CB}
\DeclareTextAccent{\capitalhungarumlaut}{\ant@encoding}{"F6CF}
\DeclareTextAccent{\capitalring}{\ant@encoding}{"EFF3}
\DeclareTextAccent{\capitalcaron}{\ant@encoding}{"F6CA}
\DeclareTextAccent{\capitalbreve}{\ant@encoding}{"EFEE}
\DeclareTextAccent{\capitalmacron}{\ant@encoding}{"F6D0}
\DeclareTextAccent{\capitaldotaccent}{\ant@encoding}{"EFED}
⟨/UT1⟩
```

10.22 Final operations

Finally undefine `\ant@encoding`.

```
\let\ant@encoding\@undefined
```

¡/UT1—T1—T2A—LGR¿