fixtounicode

Clea F. Rees*

v0.1.1 (SVN Rev: 11683) 2026/02/24

Abstract

fixtounicode provides interfaces for adding 'tounicode' mappings to type1 and/or TEX fonts which lack them. expl3 and LaTeX 2ε interfaces are provided, the former being designed primarily to support type1 symbol packages.

Contents

1 Usage

`fixtounicode` (*pkg.*) `\usepackage[`⟨*package options*⟩`]{`⟨*fixtounicode*⟩`}`

loads the package with ⟨*package options*⟩.

Available options:

`debug` (*opt.*) `=true|false`

Load debugging code, which generates more verbose console and log messages.

`dingbats` (*opt.*) `=true|false`

Enable support for Zapf Dingbats if compiling with luaTEX. This option is not really specific to Dingbats, but that is probably the best known case. Enabling this option when compiling with luaTEX 1.24 or later will load the Unicode code points known to pdfTEX (unless DVI output is enabled).

The option does nother if a different or older engine is used. For pdfTEX, it is unnecessary; the mappings are enabled by default. For other engines and older luaTEX, the mappings would either be silently ignored (luaTEX) or raise errors (e.g. luaTeX in DVI mode).

`\fixtounicode` `{`⟨*comma-separated list of key-values*⟩`}`

Add `tounicode` mappings for TEX/type1 fonts which lack them.

Supported keys:

`default` Default code point.

`glyphs` Comma-separated list of glyph names. For use with `unicodes`.

---

*Bug tracker: codeberg.org/cfr/nfssext/issues | Code: codeberg.org/cfr/nfssext | Mirror: github.com/cfr42/nfssext

`mappings` Key-value list of glyph to Unicode mappings. pdfTeX and LuaTeX 1.24 and later. Probably unsuitable for package code.

`pfb` Type1 font.

`tfm` TFM.

`unicodes` Comma-separated list of code points. For use with `glyphs`.

For example, the following adds mappings for three of marvosym's symbols.

```
\usepackage{fixtounicode}
\fixtounicode{%
  tfm = umvs,
  mappings = {%
    Coffeecup = 26BD,
    Radioactivity = 2615,
    Football = 2622,
  },
}
\usepackage{marvosym}
```

When the corresponding symbols are used in the document,

```
\Coffeecup\quad
\Radioactivity\quad
\Football
```

the result is a PDF with the specified Unicode code points. This means that copy-paste works correctly, provided the viewing application supports Unicode, screen-reading software has the relevant semantic data and conversions to text etc. work as expected.

Packages which need to define sets of mappings should use either `glyphs` and `unicodes` or the expl3 functions in section 2. `mappings` is intended for specific corrections or additions at the document level.

2 Programming interface

`\fixtounicode_tounicode:nnNN` `{`⟨*TFM*⟩`}{`⟨*PFB*⟩`}`⟨*sequence 1*⟩⟨*sequence 2*⟩

`\fixtounicode_tounicode:nNN` `{`⟨*font name*⟩`}`⟨*sequence 1*⟩⟨*sequence 2*⟩

where ⟨*TFM*⟩ and ⟨*PFB*⟩ are the names of the TFM and PFB without extensions, ⟨*sequence 1*⟩ is the name of a sequence variable containing glyph names and ⟨*sequence 2*⟩ is the name of a sequence variable containing the corresponding Unicode code points. `\fixtounicode_tounicode:nNN` is a convenience wrapper for the common case in which ⟨*TFM*⟩ and ⟨*PFB*⟩ are the same. In that case ⟨*font name*⟩ is their common name, without extension.

`\fixtounicode_tounicode:nnnn` `{`⟨*TFM*⟩`}{`⟨*PFB*⟩`}{`⟨*sequence 1*⟩`}{`⟨*sequence 2*⟩`}`

`\fixtounicode_tounicode:nnn` `{`⟨*font name*⟩`}{`⟨*sequence 1*⟩`}{`⟨*sequence 2*⟩`}`

Wrappers around `\fixtounicode_tounicode:nnNN` and `\fixtounicode_tounicode:nNN` which accept is comma-separated lists of glyph names and code points in place of variables. These versions simply set sequence variables from the comma-separated lists and pass their names to the underlying functions.

3 Implementation

You do not need to read the remainder of this document in order to install or use the package.

<*sty> <@@=fixtounicode>

```
\GetIdInfo $Id: fixtounicode.dtx 11683 2026-02-24 03:57:46Z cfrees $
  {Fix missing tounicode values in TeX fonts}
\ProvidesExplPackage{\ExplFileName}
  {\ExplFileDate}{v0.1.1 \ExplFileVersion}{\ExplFileDescription}
```

`debug` (*key*)
`dingbats` (*key*)

Keys.

```
\keys_define:nn { fixtounicode }
{
  debug .bool_set:N = \l__fixtounicode_debug_bool,
  debug .initial:n = false,
  debug .default:n = true,
  dingbats .bool_set:N = \l__fixtounicode_dingbats_bool,
  dingbats .default:n = true,
  dingbats .initial:n = false,
}
```

```
\ProcessKeyOptions[fixtounicode]
```

`icode_glyphtounicode_seq` (*var.*)
`xtounicode_tounicode_seq` (*var.*)
`_fixtounicode_glyphs_seq` (*var.*)
`fixtounicode_unicode_seq` (*var.*)
`_fixtounicode_engine_seq` (*var.*)
`code_luatex_has_fix_bool` (*var.*)

Variables.

```
\seq_new:N \l__fixtounicode_glyphtounicode_seq
\seq_new:N \l__fixtounicode_tounicode_seq
\seq_new:N \l__fixtounicode_glyphs_seq
\seq_new:N \l__fixtounicode_unicodes_seq
\seq_new:N \g__fixtounicode_engine_seq
\bool_new:N \g__fixtounicode_luatex_has_fix_bool
```

Format variant.

```
\cs_generate_variant:Nn \seq_set_item:Nnn { NnV }
```

`\__fixtounicode_debug:n` (*fn.*)
`\__fixtounicode_debug:N` (*fn.*)

```
\bool_if:NTF \l__fixtounicode_debug_bool
{
  \cs_new_protected:Npn \__fixtounicode_debug:n #1
  {
    \iow_term:n {
      [fixtounicode debug]:: #1
    }
  }
  \cs_new_protected:Npn \__fixtounicode_debug:N #1
  {
    \__fixtounicode_debug_aux:eV { \cs_to_str:N #1 } #1
  }
  \cs_new_protected:Npn \__fixtounicode_debug_aux:nn #1#2
  {
    \__fixtounicode_debug:n { #1 ->#2 }
  }
  \cs_generate_variant:Nn \__fixtounicode_debug_aux:nn {eV}
  \sys_if_engine_luatex:T
  {
    \lua_now:n { fixtounicode_debug = true }
  }
}{
  \cs_new_eq:NN \__fixtounicode_debug:n \use_none:n
  \cs_new_eq:NN \__fixtounicode_debug:N \use_none:n
}
```

If the LuaTeX version is less than 1.24, set `dev` to `false`; o/w `true`.

```
\sys_if_engine_luatex:T
{
  \__fixtounicode_debug:n {Engine is LuaTeX.}
  \bool_set_true:N \l__fixtounicode_luatex_has_fix_bool
  \seq_gset_split:NnV \g__fixtounicode_engine_seq {.} \c_sys_engine_version_str
  \int_compare:nNnTF { \seq_item:Nn \g__fixtounicode_engine_seq {1} } < {1}
  {
    \__fixtounicode_debug:n {
      LuaTeX version 0 or less:
       you get out too much.
       Stay home and update!
    }
    \bool_set_false:N \l__fixtounicode_luatex_has_fix_bool
  } {
    \int_compare:nNnT { \seq_item:Nn \g__fixtounicode_engine_seq {2} } < {24}
    {
      \__fixtounicode_debug:n {
        LuaTeX minor version less than 24.
         I'll do my best,but binary too old for full support.
      }
      \bool_set_false:N \l__fixtounicode_luatex_has_fix_bool
    }
  }
  \__fixtounicode_debug:N \l__fixtounicode_luatex_has_fix_bool
}
```

This is not good at all . . . .

```
\sys_ensure_backend:
```

`\pdfglyphtounicode` LuaTeX only, from the manual.

```
\bool_lazy_and:nnT
{ \sys_if_engine_luatex_p: }
{ \sys_if_output_pdf_p: }
{
  \protected\def\pdfglyphtounicode {\pdfextension glyphtounicode }
  \lua_now:n { pdf.setgentounicode(1) }
  \bool_if:NT \l__fixtounicode_luatex_has_fix_bool { \input glyphtounicode.tex\relax }
  \bool_if:NT \l__fixtounicode_dingbats_bool
  {
    \input glyphtounicode.tex \relax
  }
}

\msg_new:nnn { fixtounicode } { mapping }
{
  \msg_info_text:n { fixtounicode } ::
  Mapping #1 ->#2 \msg_line_context:
}
\msg_new:nnn { fixtounicode } {  limitations }
{
  \msg_warning_text:n { fixtounicode } ::
  Sorry,use of #1 is not supported on #2 \msg_line_context:.
  #3
}
\msg_new:nnn { fixtounicode } { file-awol }
{
  \msg_error_text:n { fixtounicode } ::
  #1 not found on \msg_line_context:
}
```

`xtounicode_tounicode:nnNN` (*fn.*)
`ixtounicode_tounicode:nNN` (*fn.*)
`unicode_tounicode_pair:nn` (*fn.*)
`xtounicode_tounicode:VVNN` (*fn.*)

Engine-specific functions for mappings. If we're using pdfTEX, things are straightforward: we simply use the `tfm:` syntax for the primitive `\pdfglyphtounicode`. If we're using LuaTEX, things are complicated. For newer releases (2026), the macro `\pdfglyphtounicode` defined above will accept `tfm:`. Otherwise, we could apply values directly to the `tfm`, but that will only work for the newer releases (which don't need it), so we apply them to the type1 `pfb` for now, using a workaround from Max Chernoff.

```
\cs_new_protected_nopar:Npn \__fixtounicode_tounicode_pair:nn #1#2 {}
\bool_if:nT
{
  \bool_lazy_and_p:nn
  {
    \bool_lazy_or_p:nn {\sys_if_engine_pdftex_p:}
    {
      \bool_lazy_and_p:nn
      {\sys_if_engine_luatex_p:}
      {\l__fixtounicode_luatex_has_fix_bool}
    }
  } {\sys_if_output_pdf_p:}
} {
  \__fixtounicode_debug:n {
    TFM specific tounicode mappings supported.
     Enabling support for glyph names.
  }
  \cs_new_protected_nopar:Npn \__fixtounicode_tounicode:nNN #1#2#3
  {
    \cs_set_nopar:Npn \__fixtounicode_tounicode_pair:nn ##1##2
    {
```

TFM-specific mapping.

pdfTEX manual page 33.

```
      \pdfglyphtounicode { tfm:#1/##1 } { ##2 }
      \msg_info:nnnn {fixtounicode} {mapping} {tfm:#1/##1} {##2}
    }
    \seq_set_eq:NN \l__fixtounicode_glyphtounicode_seq #2
    \seq_set_eq:NN \l__fixtounicode_tounicode_seq #3
    \seq_map_pairwise_function:NNN \l__fixtounicode_glyphtounicode_seq
      \l__fixtounicode_tounicode_seq \__fixtounicode_tounicode_pair:nn
  }
  \cs_new_protected_nopar:Npn \__fixtounicode_tounicode:nnNN #1#2#3#4
  {
    \__fixtounicode_tounicode:nNN { #1 } #3 #4
  }
}
```

For LuaTEX without the `dev` option, we use the workaround mentioned above. This deals with a limitation in the engine (lack of support for the `tfm:` syntax in `\pdfextension tounicode`) and a bug (failure to recognise any `tounicode` mappings specified for `tfm`s).

```
\bool_lazy_all:nT
{
  {\sys_if_engine_luatex_p:}
  {! \l__fixtounicode_luatex_has_fix_bool}
  {\sys_if_output_pdf_p:}
} {
  \__fixtounicode_debug:n {
    TFM specific tounicode mappings unsupported: LuaHBTeX too old.
    Enabling workaround.
    Note glyph names unsupported.
    Support requires indices.
  }
  \lua_load_module:n { fixtounicode }
  \cs_set_nopar:Npn \__fixtounicode_tounicode_pair:nn #1#2
  {
    \lua_now:n {
      table.insert(fixtounicodeTab,tonumber("#2",16))
    }
  }
  \cs_new_protected_nopar:Npn \__fixtounicode_tounicode:nnNN #1#2#3#4
  {
    \__fixtounicode_debug:n {
      Trying to add tounicode mappings for old LuaTeX.
      Fingers crossed!
    }
    \__fixtounicode_debug:n {TFM: #1; PFB: #2;}
    \__fixtounicode_debug:N #3
    \__fixtounicode_debug:N #4
    \lua_now:n { fixtounicodeTab = {} }
    \seq_set_eq:NN \l__fixtounicode_glyphtounicode_seq #3
    \seq_set_eq:NN \l__fixtounicode_tounicode_seq #4
    \seq_map_pairwise_function:NNN \l__fixtounicode_glyphtounicode_seq
      \l__fixtounicode_tounicode_seq \__fixtounicode_tounicode_pair:nn
    \lua_now:e {
      fixtounicode.tounicodes("#1","#2",fixtounicodeTab)
    }
  }
  \cs_new_protected_nopar:Npn \__fixtounicode_tounicode:nNN #1#2#3
  {
    \__fixtounicode_tounicode:nnNN { #1 } { #1 } #2 #3
  }
}
```

On all other engines, the functions are noop.

```
\sys_if_output_dvi:T
{
  \__fixtounicode_debug:n {
    tounicode mappings unsupported for DVI.
    Installing noop functions.
  }
  \cs_new_eq:NN \__fixtounicode_tounicode:nnNN \use_none:nnnn
  \cs_new_eq:NN \__fixtounicode_tounicode:nNN \use_none:nnn
}
\cs_generate_variant:Nn \__fixtounicode_tounicode:nnNN { VVNN }
```

`xtounicode_tounicode:nnNN` (*fn.*)
`ixtounicode_tounicode:nNN` (*fn.*)
`xtounicode_tounicode:nnnn` (*fn.*)
`ixtounicode_tounicode:nnn` (*fn.*)

Public expl3.

```
\cs_new_eq:NN \fixtounicode_tounicode:nnNN \__fixtounicode_tounicode:nnNN
\cs_new_eq:NN \fixtounicode_tounicode:nNN \__fixtounicode_tounicode:nNN
\cs_new_protected_nopar:Npn \fixtounicode_tounicode:nnnn #1#2#3#4
{
  \seq_set_split:Nnn \l__fixtounicode_glyphs_seq { , } { #3 }
  \seq_set_split:Nnn \l__fixtounicode_unicodes_seq { , } { #4 }
  \__fixtounicode_tounicode:nnNN { #1 } { #2 }
    \l__fixtounicode_glyphs_seq \l__fixtounicode_unicodes_seq
}
\cs_new_protected_nopar:Npn \fixtounicode_tounicode:nnn #1#2#3
{
  \seq_set_split:Nnn \l__fixtounicode_glyphs_seq { , } { #2 }
  \seq_set_split:Nnn \l__fixtounicode_unicodes_seq { , } { #3 }
  \__fixtounicode_tounicode:nnNN { #1 } { #1 }
    \l__fixtounicode_glyphs_seq \l__fixtounicode_unicodes_seq
}
```

`default` (*key*)
`glyphs` (*key*)
`mappings` (*key*)
`pfb` (*key*)
`tfm` (*key*)
`unicodes` (*key*)

Keys.

```
\keys_define:nn { fixtounicode }
{
  default .tl_set:N = \l__fixtounicode_default_tl,
  default .initial:n = 2FFFF,
  default .default:V = \c_empty_tl,
  glyphs .clist_set:N = \l__fixtounicode_glyphs_clist,
  glyphs .value_required:n = true,
  mappings .code:n = {
    \prop_put_from_keyval:Nn \l__fixtounicode_mappings_prop { #1 }
  },

  pfb .tl_set:N = \l__fixtounicode_pfb_tl,
  pfb .initial:V = \c_empty_tl,
  tfm .tl_set:N = \l__fixtounicode_tfm_tl,
  tfm .initial:V = \c_empty_tl,
  unicodes .clist_set:N = \l__fixtounicode_unicodes_clist,
  unicodes .value_required:n = true,
}
```

```
\prop_new:N \l__fixtounicode_mappings_prop
```

`__fixtounicode_tounicode:` (*fn.*)

Generic interface.

```
\cs_new_protected:Npn \__fixtounicode_tounicode:
{
  \tl_if_empty:NT \l__fixtounicode_tfm_tl
  {
    \tl_if_empty:NTF \l__fixtounicode_pfb_tl
    {
      \msg_error:nnnV {fixtounicode} {file-awol} \l__fixtounicode_pfb_tl
    }{
      \tl_set_eq:NN \l__fixtounicode_tfm_tl \l__fixtounicode_pfb_tl
    }
  }
  \tl_if_empty:NT \l__fixtounicode_pfb_tl
  {
    \tl_set_eq:NN \l__fixtounicode_pfb_tl \l__fixtounicode_tfm_tl
  }
  \seq_set_from_clist:NN \l__fixtounicode_glyphs_seq \l__fixtounicode_glyphs_clist
  \seq_set_from_clist:NN \l__fixtounicode_unicodes_seq \l__fixtounicode_unicodes_clist
  \seq_map_indexed_inline:Nn \l__fixtounicode_unicodes_seq
  {
    \tl_if_eq:VnTF \c_empty_tl { ##2 }
    {
      \seq_set_item:NnV \l__fixtounicode_unicodes_seq { ##1 } \l__fixtounicode_default_tl
    } {
      \tl_if_eq:nnT { ##2 } { 0 }
      {
        \seq_set_item:NnV \l__fixtounicode_unicodes_seq { ##1 } \l__fixtounicode_default_tl
      }
    }
  }
```

We now use any key-value mappings set *via* the `mappings` key. This is intended for cases where only a few mappings are needed from a particular font. Font support packages providing mappings for symbol fonts should preferably use the public `expl3` functions or, failing that, the keys `glyphs` and `unicodes` as I expect the implementation of `mappings` to be significantly slower.

Note that this method is **NOT** currently supported on LuaTEX. To test on LuaTEX, install a development binary, (re)generate appropriate formats and load this package with the `dev` option. Note the binary should probably **not** be used for real documents.

For TEX Live, the 2026 pretest contains a version of LuaTEX which supports this functionality, so it is no longer necessary to download a separate binary or regenerate formats.

On pdfTEX this method should work just fine.

```
  \prop_if_empty:NF \l__fixtounicode_mappings_prop
  {
    \bool_lazy_and:nnTF
    { \sys_if_engine_luatex_p: }
    { ! \l__fixtounicode_luatex_has_fix_bool }
    {
      \msg_warning:nnnnn { fixtounicode } { limitations }
      { key   mappings }
      { this version of LuaTeX }
      { Please update your TeX distribution.
        If you cannot update,use the keys glyphs and unicodes
          or equivalent expl3 functions.
        Alternatively,compile with pdfTeX.
      }
    }{
      \prop_map_inline:Nn \l__fixtounicode_mappings_prop
      {
        \seq_put_right:Nn \l__fixtounicode_glyphs_seq { ##1 }
        \seq_put_right:Nn \l__fixtounicode_unicodes_seq { ##2 }
      }
    }
  }

  \__fixtounicode_tounicode:VVNN \l__fixtounicode_tfm_tl \l__fixtounicode_pfb_tl
    \l__fixtounicode_glyphs_seq \l__fixtounicode_unicodes_seq
}
```

`\fixtounicode` 2e syntax.

```
\NewDocumentCommand \fixtounicode { +m }
{
  \group_begin:
    \keys_set:nn { fixtounicode } { #1 }
    \__fixtounicode_tounicode:
  \group_end:
}
```

</sty>

Change History

v0.1.1

General: First public release. 1

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.