

The `ltnicode.dtx` file* for use with L^AT_EX 2_ε

The L^AT_EX3 Project

July 1, 2015

This file is maintained by the L^AT_EX Project team.
Bug reports can be opened (category `latex`) at
<http://latex-project.org/bugs.html>.

This script extracts data from the Unicode Consortium files `UnicodeData.txt`, `EastAsianWidth.txt` and `LineBreak.txt` to be used for setting up L^AT_EX 2_ε (or plain T_EX) with sane default settings when using the XeT_EX and LuaT_EX engines. Details of the process are included in the code comments.

To create the extracted file, run this file in a location containing the three input data files using `pdftex`. (The code requires `\pdfmdfivesum` and the e-T_EX extensions: it could be adapted for LuaT_EX).

```
1 \script
```

1 General set up

The script is designed to work with plain T_EX and so `@` is made into a ‘letter’ using the primitive approach.

```
2 \catcode'\@=11 %
```

```
\gobble Standard utilities.
\firsttoken 3 \long\def\gobble#1{}
4 \long\def\firsttoken#1#2\relax{#1}
```

```
\storedpar A simple piece of test set up: the final line of the read file will be tokenized by TEX
as \par which can be tested by \ifx provided we have an equivalent available.
```

```
5 \def\storedpar{\par}
```

```
\return A stored ^~M for string comparisons.
```

```
6 \begingroup
7 \catcode'\^~M=12 %
8 \gdef\return{^~M}%
9 \endgroup%
```

*This file has version number v1.0i, dated 2015/06/20.

`\sourceforhex` Some parts of the code here will need to be able to convert integers to their hexadecimal equivalent. That is easiest to do for the requirements here using a modified version of some code from Appendix D of *The T_EXbook*.

```

\sethex
\dohex
\hexdigit
10 \newcount\sourceforhex
11 \def\sethex#1#2{%
12   \def#1{%
13     \sourceforhex=#2\relax
14     \ifnum\sourceforhex=0 %
15       \def#1{0}%
16     \else
17       \dohex#1%
18     \fi
19 }
20 \def\dohex#1{%
21   \begingroup
22     \count0=\sourceforhex
23     \divide\sourceforhex by 16 %
24     \ifnum\sourceforhex>0 %
25       \dohex#1%
26     \fi
27     \count2=\sourceforhex
28     \multiply\count2 by -16 %
29     \advance\count0 by\count2
30     \hexdigit#1%
31   \expandafter\endgroup
32   \expandafter\def\expandafter#1\expandafter{#1}%
33 }
34 \def\hexdigit#1{%
35   \ifnum\count0<10 %
36     \edef#1{#1\number\count0}%
37   \else
38     \advance\count0 by -10 %
39     \edef#1{#1\ifcase\count0 A\or B\or C\or D\or E\or F\fi}%
40   \fi
41 }

```

`\unicoderead, \unicodewrite` Set up the streams for data.

```

42 \newread\unicoderead
43 \newwrite\unicodewrite

```

2 Verbatim copying

`\verbatimcopy` Set up to read some material verbatim and write it to the output stream. There needs to be a dedicated ‘clean up first line’ macro, but other than that life is simple enough.

```

\endverbatimcopy
\verbatimcopy@auxii
\verbatimcopy@auxii
\verbatim@endmarker
44 \begingroup
45   \catcode'\^M=12 %
46   \gdef\verbatimcopy{%
47     \begingroup%
48       \catcode'\^M=12 %
49       \catcode'\=12 %
50       \catcode'\{=12 %

```

```

51      \catcode'\}=12 %
52      \catcode'\#=12 %
53      \catcode'\%=12 %
54      \catcode'\ =12 %
55      \endlinechar='\^M %
56      \verbatimcopy@auxi
57 }%
58  \gdef\verbatimcopy@auxi#1^M{%
59    \expandafter\verbatimcopy@auxii\gobble#1^M%
60 }%
61  \gdef\verbatimcopy@auxii#1^M{%
62    \def\temp{#1}%
63    \ifx\temp\verbatim@endmarker%
64      \expandafter\endgroup%
65    \else%
66      \ifx\temp\empty\else%
67        \immediate\write\unicodewrite{#1}%
68      \fi%
69      \expandafter\verbatimcopy@auxii%
70    \fi%
71 }%
72 \endgroup%
73 \edef\verbatim@endmarker{\expandafter\gobble\string\}
74 \edef\verbatim@endmarker{\verbatim@endmarker endverbatimcopy}

```

3 File header section

With the mechanisms set up, open the data file for writing.

```
75 \immediate\openout\unicodewrite=unicode-letters.def %
```

There are various lines that now need to go at the start of the file. First, there is some header information. Parts of it are auto-generated, so there is some interspersing of verbatim and non-verbatim parts.

```

76 \verbatimcopy
77 %% This is the file 'unicode-letters.def',
78 %% generated using the script ltunicode.dtx.
79 %%
80 %% The data here are derived from the files
81 \endverbatimcopy

```

<pre> \parseunicodedata \parseunicodedata@auxi \parseunicodedata@auxii \mdfiveinfo </pre>	<p>To ensure that there is a full audit trail for the data, we record both the reported file version (if available) and the checksum for each of the source files. This is done by reading the first line of each file and parsing for the version string and if found reading the second line for a date/time, and then ‘catching’ the entire files inside a macro to work out the checksums.</p>
---	--

```

82 \def\parseunicodedata#1{%
83   \openin\unicoderead=#1.txt %
84   \ifeof\unicoderead
85     \errmessage{Data file missing: #1.txt}%
86   \fi
87   \immediate\write\unicodewrite{%
88     \expandafter\gobble\string\% \expandafter\gobble\string\%
89     - #1.txt

```

```

90 }%
91 \readline\unicoderead to \unicodedataline
92 \edef\unicodedataline{\unicodedataline\detokenize{-.txt}}%
93 \expandafter\parseunicodedata@auxi\unicodedataline\relax{#1}%
94 }
95 \begingroup
96 \catcode'\T=12 %
97 \catcode'\X=12 %
98 \lowercase{%
99   \endgroup
100   \def\parseunicodedata@auxi#1-#2.TXT#3\relax#4}%
101 {%
102   \ifx\relax#2\relax
103     \else
104       \readline\unicoderead to \unicodedataline
105       \expandafter\parseunicodedata@auxii\unicodedataline\relax
106     \fi
107   \closein\unicoderead
108   \begingroup
109     \everyeof{\noexpand}%
110     \catcode'\#=12 %
111     \edef\mdfiveinfo{\input#4.txt\space}%
112     \expandafter\endgroup
113     \expandafter\def\expandafter\mdfiveinfo\expandafter{\mdfiveinfo}%
114     \immediate\write\unicodewrite{%
115       \expandafter\gobble\string\%\expandafter\gobble\string\%
116       \space\space
117       \ifx\relax#2\relax
118         \else
119           Version #2 dated \temp^^J%
120           \expandafter\gobble\string\%\expandafter\gobble\string\%
121           \space\space
122         \fi
123         MD5 sum \pdfmdfivesum\expandafter{\mdfiveinfo}%
124       }%
125   }
126   \def\parseunicodedata@auxii#1: #2, #3 #4\relax{%
127     \def\temp{#2, #3}%
128   }
129   \parseunicodedata{UnicodeData}
130   \parseunicodedata{EastAsianWidth}
131   \parseunicodedata{LineBreak}

132 \verbatimcopy
133 %% which are maintained by the Unicode Consortium.
134 %%
135 \endverbatimcopy

  Automatically include the current date.

136 \immediate\write\unicodewrite{%
137   \expandafter\gobble\string\%\expandafter\gobble\string\%
138   Generated on \the\year
139   -\ifnum\month>9 \else 0\fi \the\month
140   -\ifnum\day>9 \else 0\fi \the\day.
141 }

```

Back to simple text copying

```

142 \verbatimcopy
143 %%
144 %% Copyright 2014-2015
145 %% The LaTeX3 Project and any individual authors listed elsewhere
146 %% in this file.
147 %%
148 %% This file is part of the LaTeX base system.
149 %% -----
150 %%
151 %% It may be distributed and/or modified under the
152 %% conditions of the LaTeX Project Public License, either version 1.3c
153 %% of this license or (at your option) any later version.
154 %% The latest version of this license is in
155 %%   http://www.latex-project.org/lppl.txt
156 %% and version 1.3c or later is part of all distributions of LaTeX
157 %% version 2005/12/01 or later.
158 %%
159 %% This file has the LPPL maintenance status "maintained".
160 %%
161 %% The list of all files belonging to the LaTeX base distribution is
162 %% given in the file 'manifest.txt'. See also 'legal.txt' for additional
163 %% information.
164 \endverbatimcopy

```

4 Unicode character data

```

\parseunicodedata The first step of parsing a line of data is to check that it's not come from a blank
\parseunicodedata@auxi in the source, which will have been tokenized as \par. Assuming that is not the
\parseunicodedata@auxii case, there are lots of data items separated by ;. Of those, only a few are needed
\parseunicodedata@auxiii so they are picked out and everything else is dropped. There is one complication:
\parseunicodedata@auxiv there are a few cases in the data file of ranges which are marked by the descriptor
\parseunicodedata@auxv First and a matching Last. A separate routine is used to handle these cases.
\parseunicodedata@auxvi
165 \def\parseunicodedata#1{%
166   \ifx#1\storedpar
167   \else
168     \expandafter\parseunicodedata@auxi#1\relax
169   \fi
170 }
171 \def\parseunicodedata@auxi#1;#2;#3;#4;#5;#6;#7;#8;#9;{%
172   \parseunicodedata@auxii#1;#3;#2 First>\relax
173 }
174 \def\parseunicodedata@auxii#1;#2;#3 First>#4\relax{%
175   \ifx\relax#4\relax
176     \expandafter\parseunicodedata@auxiii
177   \else
178     \expandafter\parseunicodedata@auxv
179   \fi
180   #1;#2;%
181 }
182 \def\parseunicodedata@auxiii#1;#2;#3;#4;#5;#6;#7;#8\relax{%
183   \parseunicodedata@auxiv{#1}{#2}{#6}{#7}%

```

184 }

At this stage we have only four pieces of data

1. The code value
2. The general class
3. The uppercase mapping
4. The lowercase mapping

where both one or both of the last two may be empty. Everything here could be done in a single conditional within a `\write`, but that would be tricky to follow. Instead, a series of defined auxiliaries are used to show the flow. Notice that combining marks are treated as letters here (the second ‘letter’ test).

```
185 \def\parseunicodedata@auxiv#1#2#3#4{%
186   \if L\firsttoken#2?\relax
187   \expandafter\unicodeletter
188   \else
189   \if M\firsttoken#2?\relax
190   \expandafter\expandafter\expandafter\unicodeletter
191   \else
192   \expandafter\expandafter\expandafter\unicodenonletter
193   \fi
194   \fi
195   {#1}{#3}{#4}%
196 }
```

In the case where the first code point for a range was found, we assume the next line is the last code point (it always is). It’s then a question of checking if the range is a set of letters or not, and if so going through them all and adding to the data file.

```
197 \def\parseunicodedata@auxv#1;#2;#3\relax{%
198   \read\unicoderead to \unicodedataline
199   \expandafter\parseunicodedata@auxvi\unicodedataline\relax#1;#2\relax
200 }
201 \def\parseunicodedata@auxvi#1;#2\relax#3;#4\relax{%
202   \if L\firsttoken#4?\relax
203   \count@="#3 %
204   \begingroup
205   \loop
206     \unless\ifnum\count@>"#1 %
207     \sethex\temp{\count@}%
208     \unicodeletter\temp\temp\temp
209     \advance\count@\@ne
210   \repeat
211   \endgroup
212   \fi
213 }
```

codeletter, \unicodenonletter For ‘letters’, we always want to write the data to file, and the only question here is if the character has case mappings or these point back to the character itself.

```
124 \def\unicodeletter#1#2#3{%
125   \writeunicodedata\L{#1}{#2}{#3}%
126 }
```

Cased non-letters can also exist: they can be detected as they have at least one case mapping. Write these in much the same way as letters.

```

217 \def\unicodenonletter#1#2#3{%
218   \ifx\relax#2#3\relax
219   \else
220     \writeunicodedata\C{#1}{#2}{#3}%
221   \fi
222 }

```

Actually write the data. In all cases both upper- and lower-case mappings are given, so there is a need to test that both were actually available and if not set up to do nothing.

```

223 \def\writeunicodedata#1#2#3#4{%
224   \immediate\write\unicodewrite{%
225     \space\space
226     \string#1\space
227     #2 %
228     \ifx\relax#3\relax
229       #2 %
230     \else
231       #3 %
232     \fi
233     \ifx\relax#4\relax
234       #2 %
235     \else
236       #4 %
237     \fi
238     \expandafter\gobble\string\%
239   }%
240 }

```

There is now a lead-in section which creates the macros which take the processed data and do the code assignments. Everything is done within a group so that there is no need to worry about names.

```

241 \verbatimcopy
242 \begingroup
243 \endverbatimcopy

```

Cased non-letters simply need to have the case mappings set. For letters, there are a few things to sort out. First, the case mappings are defined as for non-letters. Category code is then set to 11 before a check to see if this is an upper case letter. If it is then the `\sfcode` is set to 999. Finally there is a need to deal with Unicode math codes, where base plane letters are class 7 but supplementary plane letters are class 1. Older versions of XeTeX used a different name here: easy to pick up as we know that this primitive must be defined in some way. There is also an issue with the supplementary plane and older XeTeX versions, which is dealt with using a check at run time.

```

244 \verbatimcopy
245   \def\C#1 #2 #3 {%
246     \XeTeXcheck{#1}%
247     \global\uccode"#1="#2 %
248     \global\lccode"#1="#3 %
249   }

```

```

250 \def\L#1 #2 #3 {%
251   \C #1 #2 #3 %
252   \global\catcode"#1=11 %
253   \ifnum"#1="#3 %
254   \else
255     \global\sfcodes"#1=999 %
256   \fi
257   \ifnum"#1<"10000 %
258     \global\Umathcode"#1="7"01"#1 %
259   \else
260     \global\Umathcode"#1="0"01"#1 %
261   \fi
262 }
263 \ifx\Umathcode\undefined
264   \let\Umathcode\XeTeXmathcode
265 \fi
266 \def\XeTeXcheck#1{%
267   \ifx\XeTeXversion\undefined
268   \else
269     \def\XeTeXcheck.#1.#2-#3\relax{#1}
270     \ifnum\expandafter\XeTeXcheck\XeTeXrevision.-\relax>996 %
271     \def\XeTeXcheck#1{%
272       \else
273         \def\XeTeXcheck#1{%
274           \ifnum"#1>"FFFF %
275             \long\def\XeTeXcheck##1\endgroup{\endgroup}
276             \expandafter\XeTeXcheck
277           \fi
278         }
279       \fi
280     \fi
281 \endverbatimcopy

Read the data and write the resulting code assignments to the file.

282 \openin\unicoderead=UnicodeData.txt %
283 \loop\unless\ifeof\unicoderead
284   \read\unicoderead to \unicodedataline
285   \parseunicodedata\unicodedataline
286 \repeat

End the group for setting character codes and assign a couple of special cases.

287 \verbatimcopy
288 \endgroup
289 \global\sfcodes"2019=0 %
290 \global\sfcodes"201D=0 %
291 \endverbatimcopy

```

5 XeTeX Character classes

The XeTeX engine includes the concept of character classes, which allow insertion of tokens into the input stream at defined boundaries. Setting up this data requires a two-part process as the information is split over two input files.

The parsing system is redefined to parse a detokenized input line which may be a

```

\parseunicodedata
\parseunicodedata@auxi
\parseunicodedata@auxii

```


comment starting with #. Assuming that is not the case, the data line will start with a code point potentially forming part of a range. The range is extracted and the width stored for each code point.

```

292 \def\parseunicodedata#1{%
293   \ifx#1\return
294   \else
295     \if\expandafter\gobble\string\#\expandafter\firsttoken#1?\relax
296     \else
297       \expandafter\parseunicodedata@auxi#1\relax
298     \fi
299   \fi
300 }
301 \def\parseunicodedata@auxi#1;#2 #3\relax{%
302   \parseunicodedata@auxii#1...\relax{#2}%
303 }
304 \def\parseunicodedata@auxii#1..#2..#3\relax#4{%
305   \expandafter\gdef\csname EAW@#1\endcsname{#4}%
306   \ifx\relax#2\relax
307   \else
308     \count@="#1 %
309     \begingroup
310     \loop
311       \ifnum\count@<"#2 %
312         \advance\count@\@ne
313         \sethex\temp{\count@}%
314         \expandafter\gdef\csname EAW@\temp\endcsname{#4}%
315       \repeat
316     \endgroup
317   \fi
318 }

```

With the right parser in place, read the data file.

```

319 \openin\unicoderead=EastAsianWidth.txt %
320 \loop\unless\ifeof\unicoderead
321   \readline\unicoderead to \unicodedataline
322   \parseunicodedata\unicodedataline
323 \repeat

```

\parseunicodedata@auxii The final file to read, LineBreak.txt, uses the same format as
 \parseunicodedata@auxiii EastAsianWidth.txt. As such, only the final parts of the parser have to be
 \parseunicodedata@auxiv redefined.

```

\ID 324 \def\parseunicodedata@auxii#1..#2..#3\relax#4{%
\OP 325   \parseunicodedata@auxiii{#1}{#4}%
\CL 326   \ifx\relax#2\relax
\EX 327   \else
\IS 328     \count@="#1 %
\NS 329     \begingroup
\CM 330     \loop
331       \ifnum\count@<"#2 %
332         \advance\count@\@ne
333         \sethex\temp{\count@}%
334         \expandafter\parseunicodedata@auxiii\expandafter{\temp}{#4}%
335       \repeat

```

```

336 \endgroup
337 \fi
338 }

```

Adding data to the processed file depends on two factors: the classification in the line-breaking file and (possibly) the width data too. Any characters of class ID (ideograph) are stored: they always need special treatment. For characters of classes OP (opener), CL (closer), EX (exclamation), IS (infix sep) and NS (non-starter) the data is stored if the character is full, half or wide width. The same is true for CM (combining marks) characters, which need to be transparent to the mechanism.

```

339 \def\parseunicodedata@auxiii#1#2{%
340 \ifcsname #2\endcsname
341 \ifnum\csname #2\endcsname=1 %
342 \parseunicodedata@auxiv{#1}{#2}%
343 \else
344 \ifnum 0%
345 \if F\csname EAW@#1\endcsname 1\fi
346 \if H\csname EAW@#1\endcsname 1\fi
347 \if W\csname EAW@#1\endcsname 1\fi
348 >0 %
349 \parseunicodedata@auxiv{#1}{#2}%
350 \fi
351 \fi
352 \fi
353 }
354 \def\parseunicodedata@auxiv#1#2{%
355 \immediate\write\unicodewrite{%
356 \space\space
357 \expandafter\string\csname #2\endcsname
358 \space
359 #1 %
360 \expandafter\gobble\string\%
361 }%
362 }
363 \def\ID{1}
364 \def\OP{2}
365 \def\CL{3}
366 \let\EX\CL
367 \let\IS\CL
368 \let\NS\CL
369 \def\CM{256}

```

Before actually reading the line breaking data file, the appropriate temporary code is added to the output. As described above, only a limited number of classes need to be covered: they are hard-coded as classes 1, 2 and 3 following the convention adopted by plain XeTeX.

```

370 \verbatimcopy
371 \begingroup
372 \ifx\XeTeXchartoks\XeTeXcharclass
373 \def\ID#1 {\global\catcode"#1=11 }
374 \def\OP#1 {}
375 \let\CL\OP
376 \let\EX\OP

```

```

377 \let\IS\OP
378 \let\NS\OP
379 \let\CM\OP
380 \else
381 \def\ID#1 {\global\XeTeXcharclass"#1=1 \global\catcode"#1=11 }
382 \def\OP#1 {\global\XeTeXcharclass"#1=2 }
383 \def\CL#1 {\global\XeTeXcharclass"#1=3 }
384 \def\EX#1 {\global\XeTeXcharclass"#1=3 }
385 \def\IS#1 {\global\XeTeXcharclass"#1=3 }
386 \def\NS#1 {\global\XeTeXcharclass"#1=3 }
387 \def\CM#1 {\global\XeTeXcharclass"#1=256 }
388 \fi
389 \endverbatimcopy

    Read the line breaking data and save to the output.
390 \openin\unicoderead=LineBreak.txt %
391 \loop\unless\ifeof\unicoderead
392 \readline\unicoderead to \unicodedataline
393 \parseunicodedata\unicodedataline
394 \repeat

    LuaTeX and older versions of XeTeX stop here: character classes are a XeTeX-
    only concept.
395 \verbatimcopy
396 \endgroup
397 \ifx\XeTeXchartoks\XeTeXcharclass
398 \expandafter\endinput
399 \fi
400 \endverbatimcopy

    Set up material to be inserted between character classes. that provided by
    plain XeTeX. Using \hskip here means the code will work with plain as well as
    LATEX 2ε.
401 \verbatimcopy
402 \gdef\xtxHanGlue{\hskip0pt plus 0.1em\relax}
403 \gdef\xtxHanSpace{\hskip0.2em plus 0.2em minus 0.1em\relax}
404 \global\XeTeXinterchartoks 0 1 = {\xtxHanSpace}
405 \global\XeTeXinterchartoks 0 2 = {\xtxHanSpace}
406 \global\XeTeXinterchartoks 0 3 = {\nobreak\xtxHanSpace}
407 \global\XeTeXinterchartoks 1 0 = {\xtxHanSpace}
408 \global\XeTeXinterchartoks 2 0 = {\nobreak\xtxHanSpace}
409 \global\XeTeXinterchartoks 3 0 = {\xtxHanSpace}
410 \global\XeTeXinterchartoks 1 1 = {\xtxHanGlue}
411 \global\XeTeXinterchartoks 1 2 = {\xtxHanGlue}
412 \global\XeTeXinterchartoks 1 3 = {\nobreak\xtxHanGlue}
413 \global\XeTeXinterchartoks 2 1 = {\nobreak\xtxHanGlue}
414 \global\XeTeXinterchartoks 2 2 = {\nobreak\xtxHanGlue}
415 \global\XeTeXinterchartoks 2 3 = {\xtxHanGlue}
416 \global\XeTeXinterchartoks 3 1 = {\xtxHanGlue}
417 \global\XeTeXinterchartoks 3 2 = {\xtxHanGlue}
418 \global\XeTeXinterchartoks 3 3 = {\nobreak\xtxHanGlue}
419 \endverbatimcopy

    Done: end the script.
420 \bye

```

421 `</script>`