The `ltunicode.dtx` file[*]
for use with LaTeX 2ε

The LaTeX3 Project

October 28, 2015

> This file is maintained by the LaTeX 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 LaTeX 2ε (or plain TeX) with sane default settings when using the XeTeX and LuaTeX 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-TeX extensions: it could be adapted for LuaTeX).

```
1 ⟨*script⟩
```

1 General set up

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

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

\gobble   Standard utilities.
\gobblethree
\firsttoken

```
3 \long\def\gobble#1{}
4 \long\def\gobblethree#1#2#3{}
5 \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.

```
6 \def\storedpar{\par}
```

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

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

---

[*]This file has version number v1.0l, dated 2015/08/10.

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

```
11 \newcount\sourceforhex
12 \def\sethex#1#2{%
13   \def#1{}%
14   \sourceforhex=#2\relax
15   \ifnum\sourceforhex=0 %
16     \def#1{0}%
17   \else
18     \dohex#1%
19   \fi
20 }
21 \def\dohex#1{%
22   \begingroup
23     \count0=\sourceforhex
24     \divide\sourceforhex by 16 %
25     \ifnum\sourceforhex>0 %
26       \dohex#1%
27     \fi
28     \count2=\sourceforhex
29     \multiply\count2 by -16 %
30     \advance\count0 by\count2
31     \hexdigit#1%
32   \expandafter\endgroup
33   \expandafter\def\expandafter#1\expandafter{#1}%
34 }
35 \def\hexdigit#1{%
36   \ifnum\count0<10 %
37     \edef#1{#1\number\count0}%
38   \else
39     \advance\count0 by -10 %
40     \edef#1{#1\ifcase\count0 A\or B\or C\or D\or E\or F\fi}%
41   \fi
42 }
```

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

```
43 \newread\unicoderead
44 \newwrite\unicodewrite
```

2   Verbatim copying

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

```
45 \begingroup
46   \catcode'\^^M=12 %
47   \gdef\verbatimcopy{%
48     \begingroup%
49       \catcode'\^^M=12 %
50       \catcode'\\=12 %
51       \catcode'\{=12 %
```

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

3   File header section

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

```
76  \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.

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

\parseunicodedata
\parseunicodedata@auxi
\parseunicodedata@auxii
\mdfiveinfo

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.

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

```
 91    }%
 92    \readline\unicoderead to \unicodedataline
 93    \edef\unicodedataline{\unicodedataline\detokenize{-.txt}}%
 94    \expandafter\parseunicodedata@auxi\unicodedataline\relax{#1}%
 95 }
 96 \begingroup
 97 \catcode`\T=12 %
 98 \catcode`\X=12 %
 99 \lowercase{%
100    \endgroup
101    \def\parseunicodedata@auxi#1-#2.TXT#3\relax#4}%
102 {%
103    \ifx\relax#2\relax
104    \else
105      \readline\unicoderead to \unicodedataline
106      \expandafter\parseunicodedata@auxii\unicodedataline\relax
107    \fi
108    \closein\unicoderead
109    \begingroup
110      \everyeof{\noexpand}%
111      \catcode`\#=12 %
112      \edef\mdfiveinfo{\input#4.txt\space}%
113    \expandafter\endgroup
114    \expandafter\def\expandafter\mdfiveinfo\expandafter{\mdfiveinfo}%
115    \immediate\write\unicodewrite{%
116      \expandafter\gobble\string\%\expandafter\gobble\string\%
117      \space\space
118      \ifx\relax#2\relax
119      \else
120        Version #2 dated \temp^^J%
121        \expandafter\gobble\string\%\expandafter\gobble\string\%
122        \space\space
123      \fi
124      MD5 sum \pdfmdfivesum\expandafter{\mdfiveinfo}%
125    }%
126 }
127 \def\parseunicodedata@auxii#1: #2, #3 #4\relax{%
128    \def\temp{#2, #3}%
129 }
130 \parseunicodedata{UnicodeData}
131 \parseunicodedata{EastAsianWidth}
132 \parseunicodedata{LineBreak}

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

Automatically include the current date.

```
137 \immediate\write\unicodewrite{%
138    \expandafter\gobble\string\%\expandafter\gobble\string\%
139    Generated on \the\year
140      -\ifnum\month>9 \else 0\fi \the\month
141      -\ifnum\day>9   \else 0\fi \the\day.
142 }
```

Back to simple text copying

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

4   Unicode character data

\parseunicodedata
\parseunicodedata@auxi
\parseunicodedata@auxii
\parseunicodedata@auxiii
\parseunicodedata@auxiv
\parseunicodedata@auxv
\parseunicodedata@auxvi

The first step of parsing a line of data is to check that it's not come from a blank in the source, which will have been tokenized as \par. Assuming that is not the case, there are lots of data items separated by ;. Of those, only a few are needed so they are picked out and everything else is dropped. There is one complication: there are a few cases in the data file of ranges which are marked by the descriptor First and a matching Last. A separate routine is used to handle these cases.

```
166 \def\parseunicodedata#1{%
167   \ifx#1\storedpar
168   \else
169     \expandafter\parseunicodedata@auxi#1\relax
170   \fi
171 }
172 \def\parseunicodedata@auxi#1;#2;#3;#4;#5;#6;#7;#8;#9;{%
173   \parseunicodedata@auxii#1;#3;#2 First>\relax
174 }
175 \def\parseunicodedata@auxii#1;#2;#3 First>#4\relax{%
176   \ifx\relax#4\relax
177     \expandafter\parseunicodedata@auxiii
178   \else
179     \expandafter\parseunicodedata@auxv
180   \fi
181     #1;#2;%
182 }
183 \def\parseunicodedata@auxiii#1;#2;#3;#4;#5;#6;#7;#8\relax{%
184   \parseunicodedata@auxiv{#1}{#2}{#6}{#7}%
```

```
185 }
```

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).

```
186 \def\parseunicodedata@auxiv#1#2#3#4{%
187   \if L\firsttoken#2?\relax
188     \expandafter\unicodeletter
189   \else
190     \if M\firsttoken#2?\relax
191       \expandafter\expandafter\expandafter\unicodeletter
192     \else
193       \expandafter\expandafter\expandafter\unicodenonletter
194     \fi
195   \fi
196     {#1}{#3}{#4}%
197 }
```

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 though them all and adding to the data file.

```
198 \def\parseunicodedata@auxv#1;#2;#3\relax{%
199   \read\unicoderead to \unicodedataline
200   \expandafter\parseunicodedata@auxvi\unicodedataline\relax#1;#2\relax
201 }
202 \def\parseunicodedata@auxvi#1;#2\relax#3;#4\relax{%
203   \if L\firsttoken#4?\relax
204     \count@="#3 %
205     \begingroup
206       \loop
207         \unless\ifnum\count@>"#1 %
208           \sethex\temp{\count@}%
209           \unicodeletter\temp\temp\temp
210           \advance\count@\@ne
211       \repeat
212     \endgroup
213   \fi
214 }
```

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. If there are no mappings or the mappings are all equivalent to the same code point then use a shorter version of the write macro.

```
215 \def\unicodeletter#1#2#3{%
216   \ifx\relax#2#3\relax
217     \writeunicodedatacompact\l{#1}%
218   \else
219     \ifnum 0%
220       \ifnum"#1="\ifx\relax#2\relax#1 \else#2 \fi\else1\fi
221       \ifnum"#1="\ifx\relax#3\relax#1 \else#3 \fi\else1\fi
222       >0 %
223       \writeunicodedatafull\L{#1}{#2}{#3}%
224     \else
225       \writeunicodedatacompact\l{#1}%
226     \fi
227   \fi
228 }
```

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, but always with a full mapping (must be the case to require the entry at all).

```
229 \def\unicodenonletter#1#2#3{%
230   \ifx\relax#2#3\relax
231   \else
232     \writeunicodedatafull\C{#1}{#2}{#3}%
233   \fi
234 }
```

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. Cases where both mappings are no-ops will already have been filtered out and are written in a shorter form: this saves a significant amount of space in the file.

```
235 \def\writeunicodedatafull#1#2#3#4{%
236   \immediate\write\unicodewrite{%
237     \space\space
238     \string#1\space
239     #2 %
240     \ifx\relax#3\relax
241       #2 %
242     \else
243       #3 %
244     \fi
245     \ifx\relax#4\relax
246       #2%
247     \else
248       #4%
249     \fi
250   }%
251 }
252 \def\writeunicodedatacompact#1#2{%
253   \immediate\write\unicodewrite{%
254     \space\space
255     \string#1\space
256     #2%
257   }%
258 }
```

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.

```
259 \verbatimcopy
260 \begingroup
261 \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.

```
262 \verbatimcopy
263   \def\C#1 #2 #3 {%
264     \XeTeXcheck{#1}%
265     \global\uccode"#1="#2 %
266     \global\lccode"#1="#3 %
267   }
268   \def\L#1 #2 #3 {%
269     \C #1 #2 #3 %
270     \global\catcode"#1=11 %
271     \ifnum"#1="#3 %
272     \else
273       \global\sfcode"#1=999 %
274     \fi
275     \ifnum"#1<"10000 %
276       \global\Umathcode"#1="7"01"#1 %
277     \else
278       \global\Umathcode"#1="0"01"#1 %
279     \fi
280   }
281   \def\l#1 {\L#1 #1 #1 }
282   \ifx\Umathcode\undefined
283     \let\Umathcode\XeTeXmathcode
284   \fi
285   \def\XeTeXcheck#1{}
286   \ifx\XeTeXversion\undefined
287   \else
288     \def\XeTeXcheck.#1.#2-#3\relax{#1}
289     \ifnum\expandafter\XeTeXcheck\XeTeXrevision.-\relax>996 %
290       \def\XeTeXcheck#1{}
291     \else
292       \def\XeTeXcheck#1{%
293         \ifnum"#1>"FFFF %
294           \long\def\XeTeXcheck##1\endgroup{\endgroup}
295           \expandafter\XeTeXcheck
296         \fi
297       }
298     \fi
```

```
299  \fi
300 \endverbatimcopy
```

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

```
301 \openin\unicoderead=UnicodeData.txt %
302 \loop\unless\ifeof\unicoderead
303   \read\unicoderead to \unicodedataline
304   \parseunicodedata\unicodedataline
305 \repeat
```

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

```
306 \verbatimcopy
307 \endgroup
308 \global\sfcode"2019=0 %
309 \global\sfcode"201D=0 %
310 \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.

\parseunicodedata
\parseunicodedata@auxi
\parseunicodedata@auxii
The parsing system is redefined to parse a detokenized input line which may be a comment starting with #. Assuming that is not the case, the data line with start with a code point potentially forming part of a range. The range is extracted and the width stored for each code point.

```
311 \def\parseunicodedata#1{%
312   \ifx#1\return
313   \else
314     \if\expandafter\gobble\string\#\expandafter\firsttoken#1?\relax
315     \else
316       \expandafter\parseunicodedata@auxi#1\relax
317     \fi
318   \fi
319 }
320 \def\parseunicodedata@auxi#1;#2 #3\relax{%
321   \parseunicodedata@auxii#1....\relax{#2}%
322 }
323 \def\parseunicodedata@auxii#1..#2..#3\relax#4{%
324   \expandafter\gdef\csname EAW@#1\endcsname{#4}%
325   \ifx\relax#2\relax
326   \else
327     \count@="#1 %
328     \begingroup
329       \loop
330         \ifnum\count@<"#2 %
331           \advance\count@\@ne
332           \sethex\temp{\count@}%
333           \expandafter\gdef\csname EAW@\temp\endcsname{#4}%
334       \repeat
335     \endgroup
336   \fi
337 }
```

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

```
338 \openin\unicoderead=EastAsianWidth.txt %
339 \loop\unless\ifeof\unicoderead
340   \readline\unicoderead to \unicodedataline
341   \parseunicodedata\unicodedataline
342 \repeat
```

\parseunicodedata@auxii
\parseunicodedata@auxiii
\parseunicodedata@auxiv
\parseunicodedata@auxv

The final file to read, `LineBreak.txt`, uses the same format as `EastAsianWidth.txt`. As such, only the final parts of the parser have to be redefined. The first stage here is to check if the line breaking class is known, and if so if it is equal to `ID` (class one).

\ID
\OP
\CL
\EX
\IS
\NS
\CM

```
343 \def\parseunicodedata@auxii#1..#2..#3\relax#4{%
344   \ifcsname #4\endcsname
345     \ifnum\csname #4\endcsname=\@ne
346       \expandafter\expandafter\expandafter\parseunicodedata@auxiii
347     \else
348       \expandafter\expandafter\expandafter\parseunicodedata@auxiv
349     \fi
350   \else
351     \expandafter\gobblethree
352   \fi
353     {#1}{#2}{#4}%
354 }
```

For ranges of class `ID`, the entire range is written to the data file as a single block: no need to check on the width data.

```
355 \def\parseunicodedata@auxiii#1#2#3{%
356   \immediate\write\unicodewrite{%
357     \space\space
358     \expandafter\string\csname #3\endcsname
359     \space
360     #1 \ifx\relax#2\relax#1\else#2\fi
361   }%
362 }
```

For other cases, loop over each code point separately. If the code point is of width F, H or W then the line breaking property is written to the data file. The earlier check means that this only happens for characters of classes `OP` (opener), `CL` (closer), `EX` (exclamation), `IS` (infix sep), `NS` (non-starter) and `CM` (combining marks) characters (the latter need to be transparent to the mechanism).

```
363 \def\parseunicodedata@auxiv#1#2#3{%
364   \parseunicodedata@auxv{#1}{#3}%
365   \ifx\relax#2\relax
366   \else
367     \count@="#1 %
368     \begingroup
369       \loop
370         \ifnum\count@<"#2 %
371           \advance\count@\@ne
372           \sethex\temp{\count@}%
373           \expandafter\parseunicodedata@auxv\expandafter{\temp}{#3}%
374       \repeat
375     \endgroup
376   \fi
```

```
377 }
378 \def\parseunicodedata@auxv#1#2{%
379   \ifnum 0%
380     \if F\csname EAW@#1\endcsname 1\fi
381     \if H\csname EAW@#1\endcsname 1\fi
382     \if W\csname EAW@#1\endcsname 1\fi
383      >0 %
384     \immediate\write\unicodewrite{%
385       \space\space
386       \expandafter\string\csname #2\endcsname
387       \space
388       #1%
389     }%
390   \fi
391 }
```

The East Asian width class mappings.

```
392 \def\ID{1}
393 \def\OP{2}
394 \def\CL{3}
395 \let\EX\CL
396 \let\IS\CL
397 \let\NS\CL
398 \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.

```
399 \verbatimcopy
400 \begingroup
401   \ifx\XeTeXchartoks\XeTeXcharclass
402     \endgroup\expandafter\endinput
403   \else
404     \def\setclass#1#2#3{%
405       \ifnum#1>#2 %
406         \expandafter\gobble
407       \else
408         \expandafter\firstofone
409       \fi
410         {%
411           \global\XeTeXcharclass#1=#3 %
412           \expandafter\setclass\expandafter
413             {\number\numexpr#1+1\relax}{#2}{#3}%
414         }%
415     }%
416     \def\gobble#1{}
417     \def\firstofone#1{#1}
418     \def\ID#1 #2 {\setclass{"#1}{"#2}{1}}
419     \def\OP#1 {\setclass{"#1}{"#1}{2}}
420     \def\CL#1 {\setclass{"#1}{"#1}{3}}
421     \def\EX#1 {\setclass{"#1}{"#1}{3}}
422     \def\IS#1 {\setclass{"#1}{"#1}{3}}
423     \def\NS#1 {\setclass{"#1}{"#1}{3}}
```

```
424     \def\CM#1 {\setclass{"#1}{"#1}{256}}
425   \fi
426 \endverbatimcopy
```

Read the line breaking data and save to the output.

```
427 \openin\unicoderead=LineBreak.txt %
428 \loop\unless\ifeof\unicoderead
429   \readline\unicoderead to \unicodedataline
430   \parseunicodedata\unicodedataline
431 \repeat
```

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 2$_\varepsilon$.

```
432 \verbatimcopy
433 \endgroup
434 \gdef\xtxHanGlue{\hskip0pt plus 0.1em\relax}
435 \gdef\xtxHanSpace{\hskip0.2em plus 0.2em minus 0.1em\relax}
436 \global\XeTeXinterchartoks 0 1 = {\xtxHanSpace}
437 \global\XeTeXinterchartoks 0 2 = {\xtxHanSpace}
438 \global\XeTeXinterchartoks 0 3 = {\nobreak\xtxHanSpace}
439 \global\XeTeXinterchartoks 1 0 = {\xtxHanSpace}
440 \global\XeTeXinterchartoks 2 0 = {\nobreak\xtxHanSpace}
441 \global\XeTeXinterchartoks 3 0 = {\xtxHanSpace}
442 \global\XeTeXinterchartoks 1 1 = {\xtxHanGlue}
443 \global\XeTeXinterchartoks 1 2 = {\xtxHanGlue}
444 \global\XeTeXinterchartoks 1 3 = {\nobreak\xtxHanGlue}
445 \global\XeTeXinterchartoks 2 1 = {\nobreak\xtxHanGlue}
446 \global\XeTeXinterchartoks 2 2 = {\nobreak\xtxHanGlue}
447 \global\XeTeXinterchartoks 2 3 = {\xtxHanGlue}
448 \global\XeTeXinterchartoks 3 1 = {\xtxHanGlue}
449 \global\XeTeXinterchartoks 3 2 = {\xtxHanGlue}
450 \global\XeTeXinterchartoks 3 3 = {\nobreak\xtxHanGlue}
451 \endverbatimcopy
```

Done: end the script.

```
452 \bye
```

```
453 ⟨/script⟩
```