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

The LaTeX3 Project

April 28, 2015

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` `\firsttoken` Standard utilities.

```
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 v1.0g, dated 2015/03/26.

\sourceforhex
\sethex
\dohex
\hexdigit

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 TEXbook*.

```
\newcount\sourceforhex
\def\sethex#1#2{%
  \def#1{}%
  \sourceforhex=#2\relax
  \ifnum\sourceforhex=0 %
    \def#1{0}%
  \else
    \dohex#1%
  \fi
}
\def\dohex#1{%
  \begingroup
    \count0=\sourceforhex
    \divide\sourceforhex by 16 %
    \ifnum\sourceforhex>0 %
      \dohex#1%
    \fi
    \count2=\sourceforhex
    \multiply\count2 by -16 %
    \advance\count0 by\count2
    \hexdigit#1%
  \expandafter\endgroup
  \expandafter\def\expandafter#1\expandafter{#1}%
}
\def\hexdigit#1{%
  \ifnum\count0<10 %
    \edef#1{#1\number\count0}%
  \else
    \advance\count0 by -10 %
    \edef#1{#1\ifcase\count0 A\or B\or C\or D\or E\or F\fi}%
  \fi
}
```

\unicoderead, \unicodewrite

Set up the streams for data.

```
\newread\unicoderead
\newwrite\unicodewrite
```

2 Verbatim copying

\verbatimcopy
\endverbatimcopy
\verbatimcopy@auxii
\verbatimcopy@auxii
\verbatim@endmarker

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.

```
\begingroup
  \catcode`\^^M=12 %
  \gdef\verbatimcopy{%
    \begingroup%
      \catcode`\^^M=12 %
      \catcode`\\=12 %
      \catcode`\{=12 %
```

```
      \catcode`\}=12 %
      \catcode`\#=12 %
      \catcode`\%=12 %
      \catcode`\ =12 %
      \endlinechar=`\^^M %
      \verbatimcopy@auxi
  }%
  \gdef\verbatimcopy@auxi#1^^M{%
    \expandafter\verbatimcopy@auxii\gobble#1^^M%
  }%
  \gdef\verbatimcopy@auxii#1^^M{%
    \def\temp{#1}%
    \ifx\temp\verbatim@endmarker%
      \expandafter\endgroup%
    \else%
      \ifx\temp\empty\else%
        \immediate\write\unicodewrite{#1}%
      \fi%
      \expandafter\verbatimcopy@auxii%
    \fi%
  }%
\endgroup%
\edef\verbatim@endmarker{\expandafter\gobble\string\\}
\edef\verbatim@endmarker{\verbatim@endmarker endverbatimcopy}
```

3 File header section

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

```
\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.

```
\verbatimcopy
%% This is the file `unicode-letters.def',
%% generated using the script ltunicode.dtx.
%%
%% The data here are derived from the files
\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.

```
\def\parseunicodedata#1{%
  \openin\unicoderead=#1.txt %
  \ifeof\unicoderead
    \errmessage{Data file missing: #1.txt}%
  \fi
  \immediate\write\unicodewrite{%
    \expandafter\gobble\string\%\expandafter\gobble\string\%
    - #1.txt
```

```
  }%
  \readline\unicoderead to \unicodedataline
  \edef\unicodedataline{\unicodedataline\detokenize{-.txt}}%
  \expandafter\parseunicodedata@auxi\unicodedataline\relax{#1}%
}
\begingroup
\catcode`\T=12 %
\catcode`\X=12 %
\lowercase{%
  \endgroup
  \def\parseunicodedata@auxi#1-#2.TXT#3\relax#4}%
{%
  \ifx\relax#2\relax
  \else
    \readline\unicoderead to \unicodedataline
    \expandafter\parseunicodedata@auxii\unicodedataline\relax
  \fi
  \closein\unicoderead
  \begingroup
    \everyeof{\noexpand}%
    \catcode`\#=12 %
    \edef\mdfiveinfo{\input#4.txt\space}%
  \expandafter\endgroup
  \expandafter\def\expandafter\mdfiveinfo\expandafter{\mdfiveinfo}%
  \immediate\write\unicodewrite{%
    \expandafter\gobble\string\%\expandafter\gobble\string\%
    \space\space
    \ifx\relax#2\relax
    \else
      Version #2 dated \temp^^J%
      \expandafter\gobble\string\%\expandafter\gobble\string\%
      \space\space
    \fi
    MD5 sum \pdfmdfivesum\expandafter{\mdfiveinfo}%
  }%
}
\def\parseunicodedata@auxii#1: #2, #3 #4\relax{%
  \def\temp{#2, #3}%
}
\parseunicodedata{UnicodeData}
\parseunicodedata{EastAsianWidth}
\parseunicodedata{LineBreak}

\verbatimcopy
%% which are maintained by the Unicode Consortium.
%%
\endverbatimcopy
```

Automatically include the current date.

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

Back to simple text copying

```
\verbatimcopy
%%
%% Copyright 2014-2015
%% The LaTeX3 Project and any individual authors listed elsewhere
%% in this file.
%%
%% This file is part of the LaTeX base system.
%% ------------------------------------------
%%
%% It may be distributed and/or modified under the
%% conditions of the LaTeX Project Public License, either version 1.3c
%% of this license or (at your option) any later version.
%% The latest version of this license is in
%%    http://www.latex-project.org/lppl.txt
%% and version 1.3c or later is part of all distributions of LaTeX
%% version 2005/12/01 or later.
%%
%% This file has the LPPL maintenance status "maintained".
%%
%% The list of all files belonging to the LaTeX base distribution is
%% given in the file 'manifest.txt'. See also 'legal.txt' for additional
%% information.
\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.

```
\def\parseunicodedata#1{%
  \ifx#1\storedpar
  \else
    \expandafter\parseunicodedata@auxi#1\relax
  \fi
}
\def\parseunicodedata@auxi#1;#2;#3;#4;#5;#6;#7;#8;#9;{%
  \parseunicodedata@auxii#1;#3;#2 First>\relax
}
\def\parseunicodedata@auxii#1;#2;#3 First>#4\relax{%
  \ifx\relax#4\relax
    \expandafter\parseunicodedata@auxiii
  \else
    \expandafter\parseunicodedata@auxv
  \fi
    #1;#2;%
}
\def\parseunicodedata@auxiii#1;#2;#3;#4;#5;#6;#7;#8\relax{%
  \parseunicodedata@auxiv{#1}{#2}{#6}{#7}%
```

```
}
```

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).

```
\def\parseunicodedata@auxiv#1#2#3#4{%
  \if L\firsttoken#2?\relax
    \expandafter\unicodeletter
  \else
    \if M\firsttoken#2?\relax
      \expandafter\expandafter\expandafter\unicodeletter
    \else
      \expandafter\expandafter\expandafter\unicodenonletter
    \fi
  \fi
    {#1}{#3}{#4}%
}
```

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.

```
\def\parseunicodedata@auxv#1;#2;#3\relax{%
  \read\unicoderead to \unicodedataline
  \expandafter\parseunicodedata@auxvi\unicodedataline\relax#1;#2\relax
}
\def\parseunicodedata@auxvi#1;#2\relax#3;#4\relax{%
  \if L\firsttoken#4?\relax
    \count@="#3 %
    \begingroup
      \loop
        \ifnum\count@<"#1 %
          \advance\count@\@ne
          \sethex\temp{\count@}%
          \unicodeletter\temp\temp\temp
      \repeat
    \endgroup
  \fi
}
```

`\unicodeletter`, `\unicodenonletter`, `\writeunicodedata`

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.

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

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.

```
\def\unicodenonletter#1#2#3{%
  \ifx\relax#2#3\relax
  \else
    \writeunicodedata\C{#1}{#2}{#3}%
  \fi
}
```

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.

```
\def\writeunicodedata#1#2#3#4{%
  \immediate\write\unicodewrite{%
    \space\space
    \string#1\space
    #2 %
    \ifx\relax#3\relax
      #2 %
    \else
      #3 %
    \fi
    \ifx\relax#4\relax
      #2 %
    \else
      #4 %
    \fi
    \expandafter\gobble\string\%
  }%
}
```

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.

```
\verbatimcopy
\begingroup
\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.

```
\verbatimcopy
  \def\C#1 #2 #3 {%
    \XeTeXcheck{#1}%
    \global\uccode"#1="#2 %
    \global\lccode"#1="#3 %
  }
```

```
  \def\L#1 #2 #3 {%
    \C #1 #2 #3 %
    \global\catcode"#1=11 %
    \ifnum"#1="#3 %
    \else
      \global\sfcode"#1=999 %
    \fi
    \ifnum"#1<"10000 %
      \global\Umathcode"#1="7"01"#1 %
    \else
      \global\Umathcode"#1="0"01"#1 %
    \fi
  }
  \ifx\Umathcode\undefined
    \let\Umathcode\XeTeXmathcode
  \fi
  \def\XeTeXcheck#1{}
  \ifx\XeTeXversion\undefined
  \else
    \def\XeTeXcheck.#1.#2-#3\relax{#1}
     \ifnum\expandafter\XeTeXcheck\XeTeXrevision.-\relax>996 %
       \def\XeTeXcheck#1{}
     \else
       \def\XeTeXcheck#1{%
          \ifnum"#1>"FFFF %
            \long\def\XeTeXcheck##1\endgroup{\endgroup}
            \expandafter\XeTeXcheck
          \fi
       }
     \fi
  \fi
\endverbatimcopy
```

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

```
\openin\unicoderead=UnicodeData.txt %
\loop\unless\ifeof\unicoderead
  \read\unicoderead to \unicodedataline
  \parseunicodedata\unicodedataline
\repeat
```

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

```
\verbatimcopy
\endgroup
\global\sfcode"2019=0 %
\global\sfcode"201D=0 %
\endverbatimcopy
```

LuaTEX and older versions of XeTEX stop here: character classes are a XeTEX-only concept.

```
\verbatimcopy
\ifx\XeTeXchartoks\XeTeXcharclass
  \expandafter\endinput
\fi
\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.

```
\def\parseunicodedata#1{%
  \ifx#1\return
  \else
    \if\expandafter\gobble\string\#\expandafter\firsttoken#1?\relax
    \else
      \expandafter\parseunicodedata@auxi#1\relax
    \fi
  \fi
}
\def\parseunicodedata@auxi#1;#2 #3\relax{%
  \parseunicodedata@auxii#1....\relax{#2}%
}
\def\parseunicodedata@auxii#1..#2..#3\relax#4{%
  \expandafter\gdef\csname EAW@#1\endcsname{#4}%
  \ifx\relax#2\relax
  \else
    \count@="#1 %
    \begingroup
      \loop
        \ifnum\count@<"#2 %
          \advance\count@\@ne
          \sethex\temp{\count@}%
          \expandafter\gdef\csname EAW@\temp\endcsname{#4}%
      \repeat
    \endgroup
  \fi
}
```

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

```
\openin\unicoderead=EastAsianWidth.txt %
\loop\unless\ifeof\unicoderead
  \readline\unicoderead to \unicodedataline
  \parseunicodedata\unicodedataline
\repeat
```

\parseunicodedata@auxii
\parseunicodedata@auxiii
\parseunicodedata@auxiv
\ID
\OP
\CL
\EX
\IS
\NS
\CM

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.

```
\def\parseunicodedata@auxii#1..#2..#3\relax#4{%
  \parseunicodedata@auxiii{#1}{#4}%
  \ifx\relax#2\relax
  \else
    \count@="#1 %
```

```
    \begingroup
      \loop
        \ifnum\count@<"#2 %
          \advance\count@\@ne
          \sethex\temp{\count@}%
          \expandafter\parseunicodedata@auxiii\expandafter{\temp}{#4}%
      \repeat
    \endgroup
  \fi
}
```

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.

```
\def\parseunicodedata@auxiii#1#2{%
  \ifcsname #2\endcsname
    \ifnum\csname #2\endcsname=1 %
      \parseunicodedata@auxiv{#1}{#2}%
    \else
      \ifnum 0%
        \if F\csname EAW@#1\endcsname 1\fi
        \if H\csname EAW@#1\endcsname 1\fi
        \if W\csname EAW@#1\endcsname 1\fi
        >0 %
        \parseunicodedata@auxiv{#1}{#2}%
      \fi
    \fi
  \fi
}
\def\parseunicodedata@auxiv#1#2{%
  \immediate\write\unicodewrite{%
    \space\space
    \expandafter\string\csname #2\endcsname
    \space
    #1 %
    \expandafter\gobble\string\%
  }%
}
\def\ID{1}
\def\OP{2}
\def\CL{3}
\let\EX\CL
\let\IS\CL
\let\NS\CL
\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.

```
\verbatimcopy
\begingroup
  \def\ID#1 {\global\XeTeXcharclass"#1=1 \global\catcode"#1=11 }
  \def\OP#1 {\global\XeTeXcharclass"#1=2 }
  \def\CL#1 {\global\XeTeXcharclass"#1=3 }
  \def\EX#1 {\global\XeTeXcharclass"#1=3 }
  \def\IS#1 {\global\XeTeXcharclass"#1=3 }
  \def\NS#1 {\global\XeTeXcharclass"#1=3 }
  \def\CM#1 {\global\XeTeXcharclass"#1=256 }
\endverbatimcopy
```

Read the line breaking data and save to the output.

```
\openin\unicoderead=LineBreak.txt %
\loop\unless\ifeof\unicoderead
  \readline\unicoderead to \unicodedataline
  \parseunicodedata\unicodedataline
\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_ε.

```
\verbatimcopy
\endgroup
\gdef\xtxHanGlue{\hskip0pt plus 0.1em\relax}
\gdef\xtxHanSpace{\hskip0.2em plus 0.2em minus 0.1em\relax}
\global\XeTeXinterchartoks 0 1 = {\xtxHanSpace}
\global\XeTeXinterchartoks 0 2 = {\xtxHanSpace}
\global\XeTeXinterchartoks 0 3 = {\nobreak\xtxHanSpace}
\global\XeTeXinterchartoks 1 0 = {\xtxHanSpace}
\global\XeTeXinterchartoks 2 0 = {\nobreak\xtxHanSpace}
\global\XeTeXinterchartoks 3 0 = {\xtxHanSpace}
\global\XeTeXinterchartoks 1 1 = {\xtxHanGlue}
\global\XeTeXinterchartoks 1 2 = {\xtxHanGlue}
\global\XeTeXinterchartoks 1 3 = {\nobreak\xtxHanGlue}
\global\XeTeXinterchartoks 2 1 = {\nobreak\xtxHanGlue}
\global\XeTeXinterchartoks 2 2 = {\nobreak\xtxHanGlue}
\global\XeTeXinterchartoks 2 3 = {\xtxHanGlue}
\global\XeTeXinterchartoks 3 1 = {\xtxHanGlue}
\global\XeTeXinterchartoks 3 2 = {\xtxHanGlue}
\global\XeTeXinterchartoks 3 3 = {\nobreak\xtxHanGlue}
\endverbatimcopy
```

Done: end the script.

```
\bye
```

⟨/script⟩