Distinguishing 8-bit characters and Japanese characters in (u)pTEX

Hironori Kitagawa

Abstract

pTEX (an extension of TeX for Japanese typesetting) uses a legacy encoding as the internal Japanese encoding, while accepting UTF-8 input. This means that pTEX does code conversion in input and output. Also, pTEX (and its Unicode extension upTEX) distinguishes 8-bit character tokens and Japanese character tokens, while this distinction disappears when tokens are processed with \string and \meaning, or printed to a file or the terminal.

These facts cause several unnatural behaviors with (u)pTEX. For example, pTEX garbles “f” (long s) to “顛” on some occasions. This paper explains these unnatural behaviors, and discusses an experiment in improvement by the author.

1 Introduction

Since TeX Live 2018, UTF-8 has been the new default input encoding in LaTeX [8]. However, with pLATEX, which is a modified version of LaTeX for the pTEX engine, the source

```latex
\documentclass{minimal}
\begin{document}
\texttt{% long s}
\end{document}
```

gives an inconsistent error message [4] (edited to fit TUGboat’s narrow columns):

```
! Package inputenc Error: Unicode character \texttt{\texttt{% long s}} not set up for use with \LaTeX.
```

Here “顛”, “f” and U+0C4CF are all different characters.

The purpose of this paper is to investigate the background of this message and propose patches to resolve this issue. This paper is based on a cancelled talk [6] in TeXConf 2019.1

In this paper, the following are assumed:

- All inputs and outputs are encoded in UTF-8.
- pTEX uses EUC-JP as the internal Japanese encoding (see Section 2.1).
- Sources are typeset in plain pTEX (ptex), unless stated otherwise by \texttt{%}.
- The notation \texttt{<AB>} describes a byte \texttt{0xab}, or a character token whose code is \texttt{0xab}.

2 Overview of pTEX

pTEX is an engine extension of TeX82 for Japanese typesetting. It can typeset Japanese documents of professional quality [9], including Japanese line breaking rules and vertical typesetting.

pTEX and pLaTeX were originally developed by the ASCII Corporation2 [1]. However, pTEX and pLaTeX in TeX Live, which are our concern, are community editions. These are currently maintained by the Japanese TeX Development Community.3 For more detail, please see the English guide for pTEX [3].

pTEX itself does not have \texttt{\vphantom} features, but there is \texttt{\vphantom}, which merges pTEX, \texttt{\vphantom} and additional primitives. Anything discussed about pTEX in this paper (besides this paragraph) also applies to \texttt{\vphantom}, so I simply write “pTEX” instead of “pTEX and \texttt{\vphantom}”. Note that the \texttt{\vphantom} format in TeX Live is produced by \texttt{\vphantom}, because recent versions of \LaTeX{} require \texttt{\vphantom} features.

2.1 Input code conversion by ptextenc

Although pTEX in TeX Live accepts UTF-8 inputs, the internal Japanese character set is limited to JIS X 0208 (JIS level 1 and 2 kanjis), which is a legacy character set before Unicode. pTEX uses Shift_JIS (Windows) or EUC-JP (other) as the internal encoding of JIS X 0208.

In pTEX and related programs, the ptextenc library [12] converts an input line to the internal encoding. pTEX’s input processor actually reads the converted result by ptextenc. A valid UTF-8 sequence does not represent a JIS X 0208 character — such as \texttt{<C5><BF>} (“f”) or \texttt{<C3><9F>} (“b”) — is converted to \texttt{\vphantom} notation, such as \texttt{\vphantom{ab}}.

On the other hand, an invalid UTF-8 sequence is converted into \texttt{\vphantom{ab}} (an undefined code point in EUC-JP) sometimes, in TeX Live 2019 or prior. In TeX Live 2020, the sequence is always converted into \texttt{\vphantom} notation.

2.2 Japanese character tokens

pTEX divides character tokens into two groups: ordinary 8-bit character tokens and Japanese character tokens. The former are not different from tokens in 8-bit engines, say, TeX82 and pdLATEX. A \texttt{\vphantom} notation sequence is always treated as an 8-bit character.

A Japanese character token is represented by its character code. In other words, although there is a \texttt{\vphantom} primitive, which is the counterpart of \texttt{\vphantom}, its information is not stored in tokens. Hence, changing \texttt{\vphantom} by users is not recommended.

---

1 TeXConf 2019 (the annual meeting of Japanese TeX users, texconf2019.tumblr.com) was canceled due to a typhoon.

2 Currently ASCII DWANGO in DWANGO Co. Ltd.

3 texjp.org/. Several GitHub repositories: [GitHub repository for ptex](https://github.com/texjporg/dwango), [GitHub repository for platex](https://github.com/texjporg/platex).
2.3 An example input

Now we look at an example. Our input line is
\[ a\texttt{c3}\texttt{c9f}\texttt{e6}\texttt{b0}\texttt{a2}\texttt{c5}\texttt{bf}\texttt{c2}\texttt{a7} \]  
(a漢§)

First, \texttt{ptexenc} converts this line into
\[ ^{a^{-}}\texttt{c3}^{9f}\texttt{b4}\texttt{c1}^{c5}^{bf}\texttt{a1}\texttt{bf} \]
which is fed to \texttt{pT\TeX}'s input processor. The final character “§” is included in JIS X 0208.

From the result above, \texttt{pT\TeX} produces tokens
\[ a_{12} \texttt{c3}_{12} \texttt{9f}_{12} \texttt{c1} \texttt{c5}_{12} \texttt{bf}_{12} \texttt{a1} \texttt{bf} § \]
where 漢 and § are Japanese character tokens. From this example, we can see that we cannot write “§” directly to output this character in a Latin font (use commands or “^{c2}\texttt{a7}”).

3 Stringization in \texttt{pT\TeX}

3.1 Overview

Names of multiletter control sequences, which include control sequences with single Japanese character name, such as \texttt{\char} or \texttt{\jobname}, are stringized, that is to say, they are stored into the string pool. Similarly, some primitives, such as \texttt{\string}, \texttt{\jobname}, \texttt{\meaning} and \texttt{\the} (almost always the case), first stringize their intermediate results into the string pool, and then retokenize these intermediate results.

Stringization of \texttt{pT\TeX} has two crucial points.

- The origin of a byte is lost in stringization. A byte sequence, for example \texttt{c5}\texttt{bf}, in the string pool may be the result of stringization of a Japanese character “顛”, or that of two 8-bit characters \texttt{c5} and \texttt{bf}.
- In retokenization, a byte sequence which represents a Japanese character in the internal encoding is \textit{always} converted to a Japanese character token. For example, \texttt{c5}\texttt{bf} is always converted to a Japanese token 頭.

These points cause unnatural behavior, namely bytes from 8-bit characters becoming garbled to Japanese character tokens. We look into several examples.

3.2 Control sequence name

Let’s begin with the following source:

\begin{verbatim}
\font\Z=ec-lmr10 \Z \% T1 encoding \edef\csname u:\endcsname{AA} \edef\csname u\endcsname{BB} \edef\ZZ\string{1} \edef\ZZ{c5\bf} \edef\csname u\endcsname{<BF>} \edef\csname u\endcsname{<C5>} \edef\csname u\endcsname{<BF>} \edef\csname u\endcsname{<C5>} \\
\cs_generate_variant:Nn \cs_to_str:N { c } \end{verbatim}

With \texttt{pT\TeX}, (1)-(3) produces the same result
\[ \texttt{BB} (\texttt{\u \head}) \]

This is because all of
\begin{verbatim}
cname u^c5^bf\endcsname \cname uf\endcsname \% f: \c5\bf in UTF-8\cname u\endcsname \cname u\endcsname \% f: \c5\bf in EUC-JP
\end{verbatim}

have the same name \texttt{c5}\texttt{bf} in \texttt{pT\TeX}, hence they are treated as the same control sequence. Applying \texttt{\string} to them, we get the same token list
\[ 12 \texttt{u}_{12} \texttt{顛} \]

This explains the error message in the introduction. “顛 (U+C4CF)” in the message is generated from

\begin{verbatim}
\expandafter\string\cname u8:\c5\bf\endcsname \edef\fuga{\c5\bf} \font\Z=ec-lmr10 \Z \% T1 encoding \edef\fuga{\c5\bf} \meaning\fuga
\end{verbatim}

The inputenc package expects that applying \texttt{\string} to the above control sequence produces
\[ 12 \texttt{u}_{12} \texttt{8}_{12} : 12 \texttt{c5}_{12} \texttt{bf}_{12} \]
but the result in \texttt{pT\TeX} is
\[ 12 \texttt{u}_{12} \texttt{8}_{12} : 12 \texttt{顛} \]

3.3 \texttt{\meaning}

The result of
\begin{verbatim}
\font\Z=ec-lmr10 \Z \% T1 encoding \def\fuga{\c5\bf\head} \meaning\fuga
\end{verbatim}
differs between plain \texttt{T\TeX} and plain \texttt{pT\TeX}:

\begin{tabular}{ll}
\texttt{plain T\TeX} macro:->Å£éąŻÅ£
\texttt{plain pT\TeX} macro:->顛顛顛
\end{tabular}

Now we look at what happened with \texttt{pT\TeX}. The definition of \texttt{\fuga} is represented by the token list
\[ \texttt{c5}_{12} \texttt{bf}_{12} \texttt{顛}_{12} \texttt{c5}_{12} \texttt{bf}_{12} \]
This gives the following string as the intermediate result of \texttt{\meaning}.
\[ \texttt{macro:->c5<bf}<c5<bf\c5<bf}\]
Retokenizing this string gives the final result
\[ \texttt{macro:->顛顛顛} \]
which we have already seen.

3.4 A tricky application

The behavior described in Section 3.2 has a tricky application: generating a Japanese character token from its code number, even in an expansion-only context. This can be constructed as follows:

\begin{verbatim}
\font\Z=ec-lmr10 \Z \% T1 encoding \input expl3-generic \ExplSyntaxOn \cs_generate_variant:Nn \cs_to_str:N { c } \end{verbatim}
Then, \texttt{pTeX} prints this token list. Since \texttt{<A1>-<FE> are printable} and \texttt{<9F> is not, the \texttt{putc2} function receives the following string, one byte per call.

\begin{verbatim}
\texttt{<C5><BF><C5><BF><C3>~9f}
\end{verbatim}

Each \texttt{<C5><BF>} is converted to “顛” by \texttt{putc2}, while the single \texttt{<C3>} remains unchanged. Hence the final result is “顛顛\texttt{<C3>~9f}, as shown.

4.3 \texttt{\message}

\texttt{\message} is similar to \texttt{\write}, but differs in that it stringizes its argument. Now consider an input line

\begin{verbatim}
\texttt{\message{~fe~f3:顛:}}
\end{verbatim}

Here 顛 (\texttt{<F0><AA><9A><B2>} in UTF-8) is a character included in JIS X 0213, but not in JIS X 0208.

The argument of \texttt{\message} is (expanded to) the following token list.

\begin{verbatim}
\texttt{<FE><F3>:<F0><AA><9A><B2>:}
\end{verbatim}

Then, this token list is stringized to

\begin{verbatim}
\texttt{<FE><F3>:<F0><AA><9A><B2>:}
\end{verbatim}

This string is “printed” by \texttt{print}; since only \texttt{<9A>} is unprintable, \texttt{putc2} receives

\begin{verbatim}
\texttt{<FE><F3>:<F0><AA><9A><B2>:}
\end{verbatim}

Now, \texttt{putc2} converts \texttt{<FE><F3>} (an undefined code point in EUC-JP) to the null character \texttt{<00>}, and \texttt{<F0><AA>} to “顛”. Hence the final result is

\begin{verbatim}
\texttt{<00>:顛~9a<B2>:}
\end{verbatim}

4.4 Controlling printability

\texttt{TEx82} and \texttt{pdfTEx} support TCX (\texttt{TEx} Character Translation) files [2], which can be used to specify which characters are printable. In fact, \texttt{cp227.tcx} is activated in (pdf)\texttt{TEx} and several other formats in \texttt{TEx} Live, to make characters 128–255 and three control characters printable. One can switch to a different TCX file at runtime. For example, only characters 32–126 are printable in

\begin{verbatim}
\texttt{latex -translate-file=empty.tcx}
\end{verbatim}

However, \texttt{pTeX} was not expected to use TCX files (no TCX files are activated in formats by \texttt{pTeX} in default). \texttt{inipTeX} can make characters printable by a TCX file, and that’s all. For example, to make characters 128–255 printable in \texttt{pTeX}, one has to make another format with appropriate option. There is no method to make an arbitrary character, say \texttt{<A0>}, unprintable when using this format.

Distinguishing 8-bit characters and Japanese characters in (u)pTEx
5 upTeX

5.1 Overview

upTeX [10,11] is a Unicode extension of pTeX by Takuji Tanaka. upTeX is (almost fully) upward-compatible with pTeX, so it is a very convenient solution for converting existing documents to Unicode with minimal changes.

In upTeX, a Japanese character token is a pair of the character code and \texttt{\kcatcode}. Furthermore, \texttt{\kcatcode} controls whether a UTF-8 sequence produces a Japanese character token or a sequence of 8-bit tokens. For example, \texttt{<E9><A1><9B>} (顛, \texttt{U+985B}) in an input line is treated as three 8-bit characters when \texttt{\kcatcode"985B=15}, and as a Japanese character otherwise.

5.2 No code conversion

Since upTeX's internal Japanese character code is Unicode (UTF-8 in the string pool), code conversion by ptexenc has no effect. Hence the inconsistent error message described in the introduction will not be issued.

5.3 Retokenization and \texttt{\kcatcode}

In upTeX, \texttt{\kcatcode} is involved in the retokenization process. Specifically, a UTF-8 sequence is converted into a Japanese character token if and only if its \texttt{\kcatcode} is not 15. This means that the result of \texttt{\meaning} of the same macro depends on \texttt{\kcatcode} settings, as in the following example.

```latex
\%#!uptex
\font\Z=ec-lmr10 \Z % T1 encoding
\% default: \texttt{\kcatcode"3042=17}
\def\fuga{^^c5^^bf
\meaning\fuga %==> macro:->ああ
\kcatcode"3042=15
\meaning\fuga %==> macro:->ああ
```

The definition of \texttt{\hoge} is represented by the token list

\texttt{<E3><B1><B2> あ,17}

Hence the intermediate result of \texttt{\meaning\hoge} is

\texttt{macro:->}<E3><B1><B2><E3><B1><B2>

However, because the \texttt{\kcatcode} of “あ” is changed, two calls of \texttt{\meaning\hoge} give different results.

We will see results of \texttt{\string} of multiletter control sequences later.

Hironori Kitagawa

6 Distinguishing bytes from 8-bit characters and those from Japanese characters

To resolve (u)pTeX’s behavior described so far, I have been developing an experimental version\footnote{\texttt{github.com/h-kitagawa/tex-jp-build/tree/printkanji_16bit}. GitHub issue: \cite{5}} of (u)pTeX, where stringization and outputting retain the origin of a byte—an 8-bit character (token) or a Japanese one. I refer to these as “experimental”, and (u)pTeX in \texttt{TeX Live} development repository as “trunk”.

The implementation approach is to extend the range of a “byte” to 0–511 (Table 1). A value between 0–255 means a byte from an 8-bit character (token), and 256–511 means a “byte” from a Japanese one.

I tested a different approach, namely using \texttt{<FF>} as a prefix to a byte 128–255 which came from an 8-bit character. But this approach caused confusion with \texttt{<FF>}, so I gave up.

6.1 \texttt{\write}

For example, consider the source from Section 4.2:

\texttt{\newwrite\OUT}
\texttt{\immediate\openout\OUT=test.dat}
\texttt{\immediate\write\OUT{顛\fuga}}
\texttt{\immediate\closeout\OUT}

with the experimental pTeX. When no TCX file is activated, putc2 receives the string

\texttt{<1C5><1BF>---c5--bf---c3--9f}

because a Japanese token 顛 sends \texttt{<1C5><1BF>} to putc2, and \texttt{<80> <FF>} are not printable. Thus the contents of the output \texttt{test.dat} are

顛\texttt{c5--bf---c3--9f}

When cp227.tcx is activated, they become

顛\texttt{f0}

because \texttt{<80> <FF>} are printable in this case.

6.2 The string pool

Since the range of a “byte” is increased to 0–511, the type of the string pool is changed to let each element store a “byte”; concretely, to a 16-bit array. For example, let’s reconsider the following source:

\texttt{\font\Z=ec-lmr10 \Z % T1 encoding}
\texttt{\def\fuga{\hoge\meaning\fuga}}

With the experimental pTeX, the intermediate result of \texttt{\meaning\fuga} is

\texttt{macro:->}<C5><BF><C5><C5><BF><C5><BF>}

Hence the result of \texttt{\meaning\fuga} is

\texttt{5}

\texttt{github.com/h-kitagawa/tex-jp-build/tree/printkanji_16bit}. GitHub issue: \cite{5}
Table 1: A “byte” in experimental (u)pTeX

<table>
<thead>
<tr>
<th>“byte” c</th>
<th>0–255</th>
<th>256–511</th>
</tr>
</thead>
<tbody>
<tr>
<td>origin</td>
<td>an 8-bit character (token)</td>
<td>a Japanese character (token)</td>
</tr>
<tr>
<td>printable characters</td>
<td>32–126 (“_”–“”)</td>
<td>all</td>
</tr>
<tr>
<td>“safe” printing of c</td>
<td>print(c)</td>
<td>print_char(c) (not print)</td>
</tr>
<tr>
<td>putc2(c,...)</td>
<td>without code conversion</td>
<td>with code conversion*</td>
</tr>
<tr>
<td>retokenization</td>
<td>an 8-bit character token c</td>
<td>a Japanese character token**</td>
</tr>
</tbody>
</table>

* Web2C’s default; can be extended by a TCX file.
** With adjacent “bytes” which are between 256–511.

I wanted to keep the modification as small and simple as possible; so I left unchanged the structure of the string pool, except for adding a “flag bit”.

6.3 Control sequence names in upTeX

In the experimental upTeX,

\csname uʃ\endcsname
\csname u顛\endcsname

are treated as different control sequences. This is because the name of the former is u<1C5><1BF>, while that of the latter is u<1C5><1BF>. This behavior seems to be natural.

However, the situation is more arguable between the experimental upTeX and the trunk upTeX. For example, let’s compare the results of (1) and (2) in the following source by both versions of upTeX.

\%#!uptex
\font\Z=ec-lmr10 \Z % T1 encoding
\def\ZZ#1{\{#1 (\string#1) \}}
\catcode"3042=15 \expandafter\def\csname \endcsname % (2)\endcsname{AA}
\catcode"3042=17 \expandafter\def\csname \endcsname {BB}
\catcode"3042=17 \expandafter\ZZ
\csname \endcsname % (1)

Results are summarized in Table 2. One may feel uneasy about both results.

trunk The results of \string for (1) and (2) differ, while they represent the same control sequence (as in Section 5.3).

experimental (1) and (2) represent different control sequences.

6.4 Input buffer(s)

I also introduced an array buffer2 as a companion array to buffer, which contains an input line. buffer2[i] plays the role of the “upper byte” of buffer[i]. Hence, when (u)pTeX considers a byte sequence buffer[i .. j] as a Japanese character, buffer2[i .. j] is set to 1. This is needed when scanning a control sequence name in order to distinguish a byte which consists a part of a Japanese character from another byte.

Suppose that the category codes of <C5> and <BF> are both 11 (letter), an input line contains \<C5><BF>\<C5><BF>\(\lowercase{\textit{顛顛}}\) (1) and pTeX is about to scan this control sequence (1). Since (p)TeX converts \--notation in a control sequence name into single characters in buffer, the contents of buffer become

\<C5><BF><C5><BF>\(\lowercase{\textit{顛顛}}\)

Thus, the control sequence (1) cannot be distinguished from \(\lowercase{\textit{顛顛}}\) so far. However, the experimental pTeX can distinguish the control sequence (1) from \(\lowercase{\textit{顛顛}}\), because the contents of buffer2 differ (see Table 3).

buffer2 is also useful in showing contexts in upTeX. For example, let’s look the following input:

\%#!uptex
\def\J{\catcode"3042=17 }
\def\L{\catcode"3042=15 }
\J \a\l \a\undefined \a\J \a
Table 2: Properties of \csname \endcsname of \TeX source in Section 6.3

<table>
<thead>
<tr>
<th><code>\kcatcode</code> `あ`</th>
<th>trunk</th>
<th>result of <code>\TEST</code></th>
<th>experimental</th>
<th>result of <code>\TEST</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) 17</td>
<td><code>&lt;E3&gt;</code> <code>&lt;B1&gt;</code> <code>&lt;B2&gt;</code> BB (あ)</td>
<td><code>&lt;E3&gt;</code> <code>&lt;B1&gt;</code> <code>&lt;B2&gt;</code> BB (あ)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2) 15</td>
<td><code>&lt;E3&gt;</code> <code>&lt;B1&gt;</code> <code>&lt;B2&gt;</code> BB (あ)</td>
<td><code>&lt;E3&gt;</code> <code>&lt;B1&gt;</code> <code>&lt;B2&gt;</code> AA (あAĆ)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Contents of `buffer` and `buffer2` when the experimental up\TeX scans control sequences in an input line

<table>
<thead>
<tr>
<th><code>\</code>顛<code>c5</code>bf<code> (\</code>顛`)</th>
<th><code>\</code>顛顛`</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>buffer</code></td>
<td><code>&lt;CF&gt;</code> <code>&lt;BF&gt;</code> <code>&lt;C5&gt;</code> <code>&lt;BF&gt;</code> <code>&lt;C5&gt;</code> <code>&lt;BF&gt;</code> <code>&lt;C5&gt;</code> <code>&lt;BF&gt;</code> <code>&lt;C5&gt;</code> <code>&lt;BF&gt;</code></td>
</tr>
<tr>
<td><code>buffer2</code></td>
<td>0 1 1 0 0 0 1 1 1 1</td>
</tr>
<tr>
<td><code>name</code></td>
<td><code>&lt;C5&gt;</code> <code>&lt;BF&gt;</code> <code>&lt;C5&gt;</code> <code>&lt;BF&gt;</code> <code>&lt;C5&gt;</code> <code>&lt;BF&gt;</code> <code>&lt;C5&gt;</code> <code>&lt;BF&gt;</code> <code>&lt;C5&gt;</code> <code>&lt;BF&gt;</code></td>
</tr>
</tbody>
</table>

With the experimental up\TeX (and no TCX file), we can know that the second “あ” is treated as three 8-bit characters from the error message. I hope this will be useful in debugging.

! Undefined control sequence.
1.3 \`あ` \`え3``81``82`\`undefined` あ\`J\` あ

The third and the final “あ” is not read by up\TeX’s input processor at the error. So they are printed as if all UTF-8 characters gave Japanese character tokens.

7 Conclusion

The primary factor of the complications discussed in this paper is that (u)p\TeX are Japanese extension of an 8-bit engine; this causes the same byte sequence can represent different things, namely a sequence of 8-bit characters (token) or Japanese characters. Although my experiment does not get rid of this factor (only ameliorates it), I hope that it is helpful.

I thank the executive committee of \TeXConf 2019, which gave me the opportunity for preparing the original talk, and the people who discussed the topics of this paper with me, especially Hironobu Yamashita, Takuji Tanaka, Takayuki Yato, and Norbert Preining.

References

  asciiwangou.github.io/ptex/index.html.


  ctan.org/pkg/ptex-manual.

JulienPalard. Inconsistent error message. github.com/texjorg/platex/issues/84.


[10] T. Tanaka. up\TeX, upLa\TeX — unicode version of p\TeX, pLa\TeX. www.t-lab.opal.ne.jp/tex/uptex_en.html.


Hironori Kitagawa