Perl & Multiple-byte Characters

Ken Lunde

CJKV Type Development

Adobe Systems Incorporated

ftp://ftp.ora.com/pub/examples/nutshell/ujip/perl/perl97.pdf

Multiple-byte Issues—Why Worry?

- Affects *all* CJKV (Chinese, Japanese, Korean, and Vietnamese) encodings—up to four bytes per character!
  - EUC-CN, ISO-2022-CN, ISO-2022-CN-EXT, HZ, and GBK for Simplified Chinese (China)
  - EUC-TW, ISO-2022-CN, ISO-2022-CN-EXT, and Big Five for Traditional Chinese (Taiwan)
  - EUC-JP, ISO-2022-JP, and Shift-JIS for Japanese
  - EUC-KR, ISO-2022-KR, Johab, and UHC for Korean
  - EUC-VN and ISO-2022-VN for Vietnamese—*not yet defined*
- Required in the context of Unicode
  - Unicode encoding is 16-bit fixed-length (aka UTF-16)
  - UTF-8 encoding is variable-length—one, two, or three bytes
- Remember: *One byte does not always equal one character*

Multiple-byte Concerns

- **Code conversion**
  - For converting data into common or different encoding
  - Required for cross-platform development
- **Data manipulation**
  - Simple text processing, such as search/replace—required for product localization
  - Related to code conversion, but not a global operation
- **Searching**
  - Simple matching—also required for product localization
- **Extensive use of regular expressions**
  - The basis for performing multiple-byte tricks through proven and related techniques such as *anchoring* and *trapping*

Code Conversion Techniques

- Algorithmic
  - Mathematical, and applied equally to *every* character
- Table-driven
  - Requires a lookup table
  - Round-trip *may* be an issue with undefined code points
- Selective
  - Convert only certain characters or certain character classes
  - Can be algorithmic or table-driven—usually table-driven
  - ***Example:*** Half- to full-width katakana—table-driven

    `ftp://ftp.ora.com/pub/examples/nutshell/ujip/perl/unkana.pl`

    ﾌｸﾞ (three characters) → フグ (two characters)
- Combination of above techniques

Algorithmic Techniques

Adobe

- Pure algorithm
  - Mathematical transformations are applied equally to *every* character
  - ***Example:*** Unicode (UTF-16) ↔ UTF-8 ↔ UTF-7
  - ***Example:*** EUC-JP ↔ ISO-2022-JP ↔ Shift-JIS
- **Normalization—identical sequence, incompatible encoding**
  - Character codes are normalized to become a continuous sequence beginning at zero
  - ***Example:*** Johab hangul ↔ Unicode hangul
    김치 (0x8BB1 0xC3A1) ↔ 김치 (0xAE40 0xCE58)

Table-driven Techniques

Adobe

- Used for conversion between incompatible encodings
  - — *Example:* Unicode ↔ other CJK encodings
  - — *Example:* UHC hangul ↔ Unicode hangul
  - — *Example:* Big Five ↔ EUC-TW (CNS 11643-1992)
  - — *Example:* Half- to full-width katakana
- Hash
  - — Simple hash-based lookup using the original (unmodified) character codes
- Zero-based table
  - — Character codes are normalized to become a continuous sequence beginning at zero
  - — Consider EUC-JP code set 1 (JIS X 0208:1997)

```
#!/usr/local/bin/perl -w

# Converting EUC-JP code set 1 to zero-based values

$ch = "\xB7\xF5"; # The kanji 剣 of JIS X 0208:1997

# Subtract 0xA1 (161) from the first byte then multiple by 94
# Subtract 0xA1 (161) from the second byte
# Add the two values to obtain zero-based value

$zeroch = ((ord(substr($ch,0,1)) - 0xA1) * 94) +
  (ord(substr($ch,1,1)) - 0xA1);
print "Zero-based value of $ch is $zeroch\n";
# $zeroch equals 2152 -- the 2,153rd character

# The following converts the zero-based value back to the original
# by reversing the effects of zero-based conversion

$ch = chr(($zeroch / 94) + 0xA1) . chr(($zeroch % 94) + 0xA1);
print "$ch\n";
# $ch again equals 0xB7F5 (剣)
```

Selective Code Conversion

- Act as filters—applied to specific characters or character classes
  - — *Example:* Simplified to traditional Chinese characters
    国 → 國
  - — *Example:* Half- to full-width katakana
- Usually table-driven, but can be algorithmic

Combination Code Conversion

- EUC-JP ↔ ISO-2022-JP ↔ Shift-JIS conversion may also involve half- to full-width katakana conversion as a method of filtering
- JConv (ANSI C) forces half- to full-width katakana conversion when converting EUC-JP or Shift-JIS to ISO-2022-JP

  `http://www.ora.com/people/authors/lunde/j_tools.html`
- The following pages provide a complete set of working Japanese code converters
  - JIS X 0208:1997, ASCII/JIS-Roman, and half-width katakana support
  - Emphasis on readability rather than efficiency—*there is more than one way to do it*

```
#!/usr/local/bin/perl -w

# ISO-2022-JP to EUC-JP

while (defined($line = <STDIN>)) {
  $line =~ s{ # JIS X 0208:1997
    \e\$[\@B]                    # ESC $ plus @ or B
    ((?:[\x21-\x7E][\x21-\x7E])+) # Two-byte characters
    \e\([BHJ]                    # ESC ( plus B, H, or J
  }{($x = $1) =~ tr/\x21-\x7E/\xA1-\xFE/, # From 7- to 8-bit
    $x
  }egx;
  $line =~ s{ # JIS X 0201-1997 half-width katakana
    \e\(I          # ESC ( I
    ([\x21-\x7E]+) # Half-width katakana
    \e\([BHJ]      # ESC ( plus B, H, or J
  }{($x = $1) =~ tr/\x21-\x7E/\xA1-\xFE/,  # From 7- to 8-bit
    ($y = $x) =~ s/([\xA1-\xFE])/\x8E$1/g, # Prefix with SS2
    $y
  }egx;
  print STDOUT $line;
}
```

```
#!/usr/local/bin/perl -w

# EUC-JP to ISO-2022-JP

while (defined($line = <STDIN>)) {
  $line =~ s{ # JIS X 0208:1997
    ((?:[\xA1-\xFE][\xA1-\xFE])+)
  }{\e\$B$1\e\(J}gx;
  $line =~ s{ # JIS X 0201-1997 half-width katakana
    ((?:\x8E[\xA0-\xDF])+)          # Half-width katakana
  }{\e\(I$1\e\(J}gx;
  $line =~ s/\x8E//g;               # Remove SS2s
  $line =~ tr/\xA1-\xFE/\x21-\x7E/; # From 8- to 7-bit
  print STDOUT $line;
}
```

```
# Some functions for Shift-JIS conversions

sub convert2sjis { # For EUC-JP and ISO-2022-JP to Shift-JIS
  my @euc = unpack("C*", $_[0]);
  my @out = ();
  while (($hi, $lo) = splice(@euc, 0, 2)) {
    $hi &= 0x7f; $lo &= 0x7f;
    push(@out, (($hi + 1) >> 1) + ($hi < 95 ? 112 : 176),
      $lo + (($hi & 1) ? ($lo > 95 ? 32 : 31) : 126));
  }
  return pack("C*", @out);
}

sub sjis2jis { # For Shift-JIS to ISO-2022-JP and EUC-JP
  my @ord = unpack("C*", $_[0]);
  for ($i = 0; $i < @ord; $i += 2) {
    $ord[$i] = (($ord[$i]-($ord[$i]<160?112:176))<<1)-
      ($ord[$i+1]<159?1:0);
    $ord[$i+1] -= ($ord[$i+1]<159?($ord[$i+1]>127?32:31):126);
  }
  return pack("C*", @ord);
}
```

```
#!/usr/local/bin/perl -w

# ISO-2022-JP or EUC-JP to Shift-JIS

while (defined($line = <STDIN>)) {
  $line =~ s{( # EUC-JP
    (?:[\xA1-\xFE][\xA1-\xFE])+| # JIS X 0208:1997
    (?:\x8E[\xA0-\xDF])+         # Half-width katakana
  )}{substr($1,0,1) eq "\x8E" ? (($x = $1) =~ s/\x8E//g, $x) :
     &convert2sjis($1)}egx;
  $line =~ s{ # Handle ISO-2022-JP
    \e\$[\@B]
    ((?:[\x21-\x7E][\x21-\x7E])+)
    \e\([BHJ]
  }{&convert2sjis($1)}egx;
  $line =~ s{ # Handle ISO-2022-JP half-width katakana
    \e\(I
    ([\x20-\x5F]+)
    \e\([BHJ]
  }{($x = $1) =~ tr/\x20-\x5F/\xA0-\xDF/, $x}egx;
  print STDOUT $line;
}
```

```
#!/usr/local/bin/perl -w

# Shift-JIS to ISO-2022-JP

while (defined($line = <STDIN>)) {
  $line =~ s{( # JIS X 0208:1997 and half-width katakana
    (?:[\x81-\x9F\xE0-\xEF][\x40-\x7E\x80-\xFC])+|
    [\xA0-\xDF]+
  )}{
    ($x=$1) !~ /^[\xA0-\xDF]/ ?
    "\e\$B" . &sjis2jis($1) . "\e\(J" :
    "\e\(I" . (($y=$x) =~ tr/\xA0-\xDF/\x20-\x5F/, $y) . "\e\(J"
  }egx;
  print STDOUT $line;
}
```

```
#!/usr/local/bin/perl -w

# Shift-JIS to EUC-JP

while (defined($line = <STDIN>)) {
  $line =~ s{( # JIS X 0208:1997 and half-width katakana
    (?:[\x81-\x9F\xE0-\xEF][\x40-\x7E\x80-\xFC])+|
    [\xA0-\xDF]+
  )}{
    ($x = $1) !~ /^[\xA0-\xDF]/ ?
    (($y = &sjis2jis($x)) =~ tr/\21-\x7E/\xA1-\xFE/, $y) :
    (($y = $x) =~ s/([\xA0-\xDF])/\x8E$1/g, $y)
  }egx;
  print STDOUT $line;
}
```

Code Conversion Pitfalls

- **No round-trip mapping for most table-driven conversions**
  - Consider Shift-JIS/EUC-JP to Unicode (UTF-16) conversion
  - Of the 8,836 code points available in Shift-JIS and EUC-JP code set 1 (JIS X 0208:1997), only 6,879 are assigned and have mappings to Unicode—the rest are converted into the *same* Unicode code point
- **Some encodings have regions that are not compatible with other related encodings**
  - ***Example:*** The Shift-JIS user-defined range (0xF040–0xFCFC) is not encoded in EUC-JP
  - ***Example:*** EUC-JP code set 3 (JIS X 0212-1990) is not encoded in Shift-JIS

Regex Techniques

Adobe

- **Multiple-byte anchoring**
  - Necessary for successful and correct matching of multiple-byte characters
- **Trapping *all* characters**
  - Otherwise, matching may occur across character boundaries
  - Necessary for selective conversion or data manipulation
- **You must specify the *complete* encoding range in order to successfully trap or anchor multiple-byte text**
  - Convenient to store the complete encoding specification in a variable for trapping and anchoring, such as `$encoding`
  - Don't forget to apply the "ox" regex modifiers when using the free-formatted encoding specifications that follow!

```
$encoding = qq<[\x00-\xFF][\x00-\xFF]>; # UCS-2

$encoding = qq< # EUC-JP
  [\x00-\x8D\x90-\xA0\xFF]|   # Code set 0 & one-byte
  \x8E[\xA0-\xDF]|           # Code set 2
  \x8F[\xA1-\xFE][\xA1-\xFE]| # Code set 3
  [\xA1-\xFE][\xA1-\xFE]      # Code set 1
>;

$encoding = qq< # EUC-TW
  [\x00-\x8D\x8F-\xA0\xFF]|             # Code set 0 & one-byte
  \x8E[\xA1-\xB0][\xA1-\xFE][\xA1-\xFE]| # Code set 2
  [\xA1-\xFE][\xA1-\xFE]                # Code set 1
>;

$encoding = qq< # EUC-KR and EUC-CN
  [\x00-\xA0\xFF]|       # Code set 0 & one-byte
  [\xA1-\xFE][\xA1-\xFE] # Code set 1
>;

$encoding = qq< # GBK
  [\x00-\x80\xFF]|              # One-byte
  [\x81-\xFE][\40-\x7E\x80-\xFE] # GBK
>;
```

```
#!/usr/local/bin/perl -w

# Multiple-byte anchoring when matching Shift-JIS–encoded text

$search = "\x8C\x95";                              # 剣
$text1  = "Text 1 \x90\x56\x8C\x95\x93\xB9"; # 新剣道
$text2  = "Text 2 \x94\x92\x8C\x8C\x95\x61"; # 白血病
$encoding = qq< # Shift-JIS encoding
  [\x00-\x80\xFD-\xFF]|       # ASCII and other one-byte
  [\xA0-\xDF]|                # Half-width katakana
  [\x81-\x9F\xE0-\xFC][\x40-\x7E\x80-\xFC] # Two-byte range
>;

print "First attempt -- no anchoring\n";
print " Matched Text1\n" if $text1 =~ /$search/o;
print " Matched Text2\n" if $text2 =~ /$search/o;

print "Second attempt -- anchoring\n";
print " Matched Text1\n" if $text1 =~ /^(?:$encoding)*?$search/ox;
print " Matched Text2\n" if $text2 =~ /^(?:$encoding)*?$search/ox;
```

Regex Techniques (Cont'd)

- Create an array that has elements with varying numbers of bytes—implements trapping
  - — Most useful for variable-length encodings
  - — Process text by iterating over the resulting array using operators such as `foreach`
  - — Using `length()` then tells you how *long* each character is

```
#!/usr/local/bin/perl -w

# This program shows how to break up multiple-byte text into
# separate array elements; it prints every character, one per
# line; two-byte characters as hexadecimal with "0x" prefix

$encoding = qq< # Shift-JIS encoding
  [\x00-\x80\xFD-\xFF]|       # ASCII and other one-byte
  [\xA0-\xDF]|                # Half-width katakana
  [\x81-\x9F\xE0-\xFC][\x40-\x7E\x80-\xFC] # Two-byte range
>;

while (defined($line = <STDIN>)) {
  @enc = $line =~ /($encoding)/gox; # One character per element
  foreach $element (@enc) {
    if (length($element) == 2) { # If two-byte character
      print STDOUT "0x" . ($x = uc unpack("H*",$element), $x);
    } else { # All others are one-byte characters
      print STDOUT "$element\n";
    }
  }
}
```

Regular Expression Pitfalls

- \W ("not a word") versus \w ("a word") in cross-platform environments—*you may get more than you bargained for*
  - The *standard* definition of \w:

    `[0-9A-Z_a-z]` (or `[\x30-\x39\x41-\x5A\x5F\x61-\x7A]`)
  - But… MacPerl's definition of \w adds the following:

    `[\x80-\x9F\xAE\xAF\xBE\xBF\xCB-\xCF\xD8\xD9\xE5-\xEF\xF1-\xF5]`
  - When encoding ranges are needed, use explicit ones
- **Specify the entire encoding range, including unused code points**
  - All code points from 0x00 through 0xFF must either represent themselves (that is, be one-byte characters), or else must be a valid first byte of a multiple-byte character

Other Useful Techniques

- Multiple-byte data is often obscured (damaged?) by Base64, URL, or quoted-printable encoding
  - HTML forms
  - E-mail
- Use the MIME module for Base64 decoding
- To decode a URL-encoded string:

```
$string =~ s/%([0-9A-Fa-f][0-9A-Fa-f])/pack("C",hex($1))/ge;
```

- To decode a quoted-printable string:

```
$string =~ s/=([0-9A-Fa-f][0-9A-Fa-f])/pack("C",hex($1))/ge;
$string =~ s/=[\x0A\x0D]+$//;
```

Advantages of Unicode

Adobe

- A *single* encoding—consider the possible encodings for the Chinese character 中 0x4E2D (“center” or “middle”)
  - Simplified Chinese = 0x5650 or 0xD6D0
  - Traditional Chinese = 0x4463, 0xC4E3, 0x8EA1C4E3, or 0xA4A4
  - Japanese = 0x4366, 0xC3E6, or 0x9286
  - Korean = 0x7169, 0xF1E9, or 0xF3E9
  - Vietnamese = 0x4A36 or 0xCAB6
- All Unicode characters—with the exception of the surrogates—are 16-bit (two bytes)
  - All characters are given equal treatment
  - No more need to deal with multiple-byte data
- Consider Java’s Unicode-based model

Advantages of Unicode (Cont'd)

- Java Version 1.1's code conversion model
  - Table-driven conversion for CJKV characters
  - Non-Unicode encodings treated as "raw data" (also called "byte sequences" or "byte arrays") that must be imported—Unicode data are considered type "char"
  - Shift-JIS → EUC-JP becomes Shift-JIS → Unicode → EUC-JP
  - Strings objects can be explicitly converted to/from Unicode
  - Input and output streams can be converted on-the-fly to/from Unicode
- Table-driven code conversion can lose data—but only undefined code points, mind you

Other Useful Information…

- "jcode.pl" library by Kazumasa Utashiro (utashiro@iij.ad.jp)

  `ftp://ftp.iij.ad.jp/pub/IIJ/dist/utashiro/perl/`

- "Unicode" module by Gisle Aas (aas@sn.no)

  `http://www.perl.com/CPAN/authors/Gisle_Aas/`

  — Supports UTF-16 (Unicode) ↔ UTF-8 algorithmic conversion

- Useful multiple-byte-capable Perl programs

  `ftp://ftp.ora.com/pub/examples/nutshell/ujip/perl/`

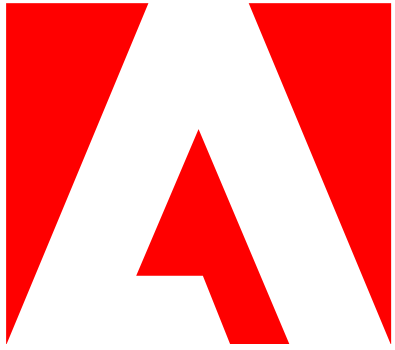
- "JPerl" by Hirofumi Watanabe (watanabe@ase.ptg.sony.co.jp)

  `http://www.perl.com/CPAN/authors/Hirofumi_Watanabe/`

  — Japanese version of Perl with Japanese-enhanced regular expressions and other Japanese-specific enhancements

  — Supports EUC-JP and Shift-JIS encodings

  — Beware of portability issues



Adobe