NAME

Encode::Unicode -- Various Unicode Transformation Formats

SYNOPSIS

```
use Encode qw/encode decode/;
$ucs2 = encode("UCS-2BE", $utf8);
$utf8 = decode("UCS-2BE", $ucs2);
```

ABSTRACT

This module implements all Character Encoding Schemes of Unicode that are officially documented by Unicode Consortium (except, of course, for UTF-8, which is a native format in perl).

http://www.unicode.org/glossary/ says:

> *Character Encoding Scheme* A character encoding form plus byte serialization. There are Seven character encoding schemes in Unicode: UTF-8, UTF-16, UTF-16BE, UTF-16LE, UTF-32 (UCS-4), UTF-32BE (UCS-4BE) and UTF-32LE (UCS-4LE), and UTF-7.
>
> Since UTF-7 is a 7-bit (re)encoded version of UTF-16BE, It is not part of Unicode's Character Encoding Scheme. It is separately implemented in Encode::Unicode::UTF7. For details see *Encode::Unicode::UTF7*.

Quick Reference

```
                Decodes from ord(N)           Encodes chr(N) to...
     octet/char BOM S.P d800-dfff  ord > 0xffff     \x{1abcd} ==
---------------+-----------------+-----------------------------
UCS-2BE 2   N   N  is bogus                 Not Available
UCS-2LE       2   N   N     bogus                 Not Available
UTF-16      2/4   Y   Y  is   S.P           S.P            BE/LE
UTF-16BE    2/4   N   Y       S.P           S.P    0xd82a,0xdfcd
UTF-16LE 2   N   Y       S.P           S.P    0x2ad8,0xcddf
UTF-32 4   Y   -  is bogus         As is            BE/LE
UTF-32BE 4   N   -     bogus         As is       0x0001abcd
UTF-32LE 4   N   -     bogus         As is       0xcdab0100
UTF-8       1-4   -   -     bogus   >= 4 octets   \xf0\x9a\af\8d
---------------+-----------------+-----------------------------
```

Size, Endianness, and BOM

You can categorize these CES by 3 criteria: size of each character, endianness, and Byte Order Mark.

by size

UCS-2 is a fixed-length encoding with each character taking 16 bits. It **does not** support *surrogate pairs*. When a surrogate pair is encountered during decode(), its place is filled with \x{FFFD} if *CHECK* is 0, or the routine croaks if *CHECK* is 1. When a character whose ord value is larger than 0xFFFF is encountered, its place is filled with \x{FFFD} if *CHECK* is 0, or the routine croaks if *CHECK* is 1.

UTF-16 is almost the same as UCS-2 but it supports *surrogate pairs*. When it encounters a high surrogate (0xD800-0xDBFF), it fetches the following low surrogate (0xDC00-0xDFFF) and `desurrogates` them to form a character. Bogus surrogates result in death. When \x{10000} or above is encountered during encode(), it `ensurrogates` them and pushes the surrogate pair to the output stream.

UTF-32 (UCS-4) is a fixed-length encoding with each character taking 32 bits. Since it is 32-bit, there is no need for *surrogate pairs*.

by endianness

The first (and now failed) goal of Unicode was to map all character repertoires into a fixed-length integer so that programmers are happy. Since each character is either a *short* or *long* in C, you have to pay attention to the endianness of each platform when you pass data to one another.

Anything marked as BE is Big Endian (or network byte order) and LE is Little Endian (aka VAX byte order). For anything not marked either BE or LE, a character called Byte Order Mark (BOM) indicating the endianness is prepended to the string.

CAVEAT: Though BOM in utf8 (\xEF\xBB\xBF) is valid, it is meaningless and as of this writing Encode suite just leave it as is (\x{FeFF}).

BOM as integer when fetched in network byte order

```
               16         32 bits/char
-------------------------
BE       0xFeFF 0x0000FeFF
LE       0xFFeF 0xFFFe0000
-------------------------
```

This modules handles the BOM as follows.

- When BE or LE is explicitly stated as the name of encoding, BOM is simply treated as a normal character (ZERO WIDTH NO-BREAK SPACE).
- When BE or LE is omitted during decode(), it checks if BOM is at the beginning of the string; if one is found, the endianness is set to what the BOM says. If no BOM is found, the routine dies.
- When BE or LE is omitted during encode(), it returns a BE-encoded string with BOM prepended. So when you want to encode a whole text file, make sure you encode() the whole text at once, not line by line or each line, not file, will have a BOM prepended.
- `UCS-2` is an exception. Unlike others, this is an alias of UCS-2BE. UCS-2 is already registered by IANA and others that way.

Surrogate Pairs

To say the least, surrogate pairs were the biggest mistake of the Unicode Consortium. But according to the late Douglas Adams in *The Hitchhiker's Guide to the Galaxy* Trilogy, `In the beginning the Universe was created. This has made a lot of people very angry and been widely regarded as a bad move`. Their mistake was not of this magnitude so let's forgive them.

(I don't dare make any comparison with Unicode Consortium and the Vogons here ;) Or, comparing Encode to Babel Fish is completely appropriate -- if you can only stick this into your ear :)

Surrogate pairs were born when the Unicode Consortium finally admitted that 16 bits were not big enough to hold all the world's character repertoires. But they already made UCS-2 16-bit. What do we do?

Back then, the range 0xD800-0xDFFF was not allocated. Let's split that range in half and use the first half to represent the `upper half of a character` and the second half to represent the `lower half of a character`. That way, you can represent 1024 * 1024 = 1048576 more characters. Now we can store character ranges up to \x{10ffff} even with 16-bit encodings. This pair of half-character is now called a *surrogate pair* and UTF-16 is the name of the encoding that embraces them.

Here is a formula to ensurrogate a Unicode character \x{10000} and above;

```
  $hi = ($uni - 0x10000) / 0x400 + 0xD800;
```

```
$lo = ($uni - 0x10000) % 0x400 + 0xDC00;
```

And to desurrogate;

```
$uni = 0x10000 + ($hi - 0xD800) * 0x400 + ($lo - 0xDC00);
```

Note this move has made \x{D800}-\x{DFFF} into a forbidden zone but perl does not prohibit the use of characters within this range. To perl, every one of \x{0000_0000} up to \x{ffff_ffff} (*) is *a character*.

```
(*) or \x{ffff_ffff_ffff_ffff} if your perl is compiled with 64-bit
integer support!
```

Error Checking

Unlike most encodings which accept various ways to handle errors, Unicode encodings simply croaks.

```
% perl -MEncode -e '$_ = "\xfe\xff\xd8\xd9\xda\xdb\0\n"' \
       -e 'Encode::from_to($_, "utf16","shift_jis", 0); print'
UTF-16:Malformed LO surrogate d8d9 at /path/to/Encode.pm line 184.
% perl -MEncode -e '$a = "BOM missing"' \
       -e ' Encode::from_to($a, "utf16", "shift_jis", 0); print'
UTF-16:Unrecognised BOM 424f at /path/to/Encode.pm line 184.
```

Unlike other encodings where mappings are not one-to-one against Unicode, UTFs are supposed to map 100% against one another. So Encode is more strict on UTFs.

Consider that "division by zero" of Encode :)

SEE ALSO

Encode, *Encode::Unicode::UTF7*, *http://www.unicode.org/glossary/*, *http://www.unicode.org/unicode/faq/utf_bom.html*,

RFC 2781 *http://rfc.net/rfc2781.html*,

The whole Unicode standard *http://www.unicode.org/unicode/uni2book/u2.html*

Ch. 15, pp. 403 of `Programming Perl (3rd Edition)` by Larry Wall, Tom Christiansen, Jon Orwant; O'Reilly & Associates; ISBN 0-596-00027-8