NAME

utf8 - Perl pragma to enable/disable UTF-8 (or UTF-EBCDIC) in source code

SYNOPSIS

```
use utf8;
no utf8;

# Convert the internal representation of a Perl scalar to/from UTF-8.

$num_octets = utf8::upgrade($string);
$success    = utf8::downgrade($string[, FAIL_OK]);

# Change each character of a Perl scalar to/from a series of
# characters that represent the UTF-8 bytes of each original character.

utf8::encode($string);  # "\x{100}"  becomes "\xc4\x80"
utf8::decode($string);  # "\xc4\x80" becomes "\x{100}"

$flag = utf8::is_utf8(STRING); # since Perl 5.8.1
$flag = utf8::valid(STRING);
```

DESCRIPTION

The `use utf8` pragma tells the Perl parser to allow UTF-8 in the program text in the current lexical scope (allow UTF-EBCDIC on EBCDIC based platforms). The `no utf8` pragma tells Perl to switch back to treating the source text as literal bytes in the current lexical scope.

Do not use this pragma for anything else than telling Perl that your script is written in UTF-8. The utility functions described below are directly usable without `use utf8;`.

Because it is not possible to reliably tell UTF-8 from native 8 bit encodings, you need either a Byte Order Mark at the beginning of your source code, or `use utf8;`, to instruct perl.

When UTF-8 becomes the standard source format, this pragma will effectively become a no-op. For convenience in what follows the term *UTF-X* is used to refer to UTF-8 on ASCII and ISO Latin based platforms and UTF-EBCDIC on EBCDIC based platforms.

See also the effects of the `-C` switch and its cousin, the `$ENV{PERL_UNICODE}`, in *perlrun*.

Enabling the `utf8` pragma has the following effect:

- Bytes in the source text that have their high-bit set will be treated as being part of a literal UTF-X sequence. This includes most literals such as identifier names, string constants, and constant regular expression patterns.

  On EBCDIC platforms characters in the Latin 1 character set are treated as being part of a literal UTF-EBCDIC character.

Note that if you have bytes with the eighth bit on in your script (for example embedded Latin-1 in your string literals), `use utf8` will be unhappy since the bytes are most probably not well-formed UTF-X. If you want to have such bytes under `use utf8`, you can disable this pragma until the end the block (or file, if at top level) by `no utf8;`.

Utility functions

The following functions are defined in the `utf8::` package by the Perl core. You do not need to say `use utf8` to use these and in fact you should not say that unless you really want to have UTF-8 source code.

* $num_octets = utf8::upgrade($string)

  Converts in-place the internal representation of the string from an octet sequence in the native encoding (Latin-1 or EBCDIC) to *UTF-X*. The logical character sequence itself is unchanged. If *$string* is already stored as *UTF-X*, then this is a no-op. Returns the number of octets necessary to represent the string as *UTF-X*. Can be used to make sure that the UTF-8 flag is on, so that `\w` or `lc()` work as Unicode on strings containing characters in the range 0x80-0xFF (on ASCII and derivatives).

  Note that this function does not handle arbitrary encodings. Therefore Encode is recommended for the general purposes; see also *Encode*.

* $success = utf8::downgrade($string[, FAIL_OK])

  Converts in-place the internal representation of the string from *UTF-X* to the equivalent octet sequence in the native encoding (Latin-1 or EBCDIC). The logical character sequence itself is unchanged. If *$string* is already stored as native 8 bit, then this is a no-op. Can be used to make sure that the UTF-8 flag is off, e.g. when you want to make sure that the substr() or length() function works with the usually faster byte algorithm.

  Fails if the original *UTF-X* sequence cannot be represented in the native 8 bit encoding. On failure dies or, if the value of `FAIL_OK` is true, returns false.

  Returns true on success.

  Note that this function does not handle arbitrary encodings. Therefore Encode is recommended for the general purposes; see also *Encode*.

* utf8::encode($string)

  Converts in-place the character sequence to the corresponding octet sequence in *UTF-X*. That is, every (possibly wide) character gets replaced with a sequence of one or more characters that represent the individual *UTF-X* bytes of the character. The UTF8 flag is turned off. Returns nothing.

```
    my $a = "\x{100}"; # $a contains one character, with ord 0x100
    utf8::encode($a);  # $a contains two characters, with ords 0xc4
and 0x80
```

  Note that this function does not handle arbitrary encodings. Therefore Encode is recommended for the general purposes; see also *Encode*.

* $success = utf8::decode($string)

  Attempts to convert in-place the octet sequence in *UTF-X* to the corresponding character sequence. That is, it replaces each sequence of characters in the string whose ords represent a valid UTF-X byte sequence, with the corresponding single character. The UTF-8 flag is turned on only if the source string contains multiple-byte *UTF-X* characters. If *$string* is invalid as *UTF-X*, returns false; otherwise returns true.

```
    my $a = "\xc4\x80"; # $a contains two characters, with ords 0xc4
and 0x80
    utf8::decode($a);   # $a contains one character, with ord 0x100
```

  Note that this function does not handle arbitrary encodings. Therefore Encode is recommended for the general purposes; see also *Encode*.

* $flag = utf8::is_utf8(STRING)

  (Since Perl 5.8.1) Test whether STRING is encoded internally in UTF-8. Functionally the same as Encode::is_utf8().

* $flag = utf8::valid(STRING)

  [INTERNAL] Test whether STRING is in a consistent state regarding UTF-8. Will return true if it is well-formed UTF-8 and has the UTF-8 flag on **or** if STRING is held as bytes (both these states are 'consistent'). Main reason for this routine is to allow Perl's testsuite to check that operations have left strings in a consistent state. You most probably want to use utf8::is_utf8() instead.

`utf8::encode` is like `utf8::upgrade`, but the UTF8 flag is cleared. See *perlunicode* for more on the UTF8 flag and the C API functions `sv_utf8_upgrade`, `sv_utf8_downgrade`, `sv_utf8_encode`, and `sv_utf8_decode`, which are wrapped by the Perl functions `utf8::upgrade`, `utf8::downgrade`, `utf8::encode` and `utf8::decode`. Also, the functions utf8::is_utf8, utf8::valid, utf8::encode, utf8::decode, utf8::upgrade, and utf8::downgrade are actually internal, and thus always available, without a `require utf8` statement.

BUGS

One can have Unicode in identifier names, but not in package/class or subroutine names. While some limited functionality towards this does exist as of Perl 5.8.0, that is more accidental than designed; use of Unicode for the said purposes is unsupported.

One reason of this unfinishedness is its (currently) inherent unportability: since both package names and subroutine names may need to be mapped to file and directory names, the Unicode capability of the filesystem becomes important-- and there unfortunately aren't portable answers.

SEE ALSO

perlunitut, *perluniintro*, *perlrun*, *bytes*, *perlunicode*