NAME

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

SYNOPSIS

```
use utf8;
no utf8;

# Convert a Perl scalar to/from UTF-8.
$num_octets = utf8::upgrade($string);
$success    = utf8::downgrade($string[, FAIL_OK]);

# Change the native bytes of a Perl scalar to/from UTF-8 bytes.
utf8::encode($string);
utf8::decode($string);

$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.

This pragma is primarily a compatibility device. Perl versions earlier than 5.6 allowed arbitrary bytes in source code, whereas in future we would like to standardize on the UTF-8 encoding for source text.

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 useful for their own purposes, but they are not really part of the "pragmatic" effect.

Until UTF-8 becomes the default format for source text, either this pragma or the *encoding* pragma should be used to recognize UTF-8 in the source. 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-8 character. 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-8. If you want to have such bytes and use utf8, you can disable utf8 until the end the block (or file, if at top level) by `no utf8;`.

If you want to automatically upgrade your 8-bit legacy bytes to UTF-8, use the *encoding* pragma instead of this pragma. For example, if you want to implicitly upgrade your ISO 8859-1 (Latin-1) bytes to UTF-8 as used in e.g. `chr()` and `\x{...}`, try this:

```
use encoding "latin-1";
my $c = chr(0xc4);
```

```
    my $x = "\x{c5}";
```

In case you are wondering: yes, `use encoding 'utf8';` works much the same as `use 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 octet sequence in the native encoding (Latin-1 or EBCDIC) to the equivalent character sequence in *UTF-X*. *$string* already encoded as characters does no harm. 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.pm* is recommended for the general purposes.

  Affected by the encoding pragma.

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

  Converts in-place the character sequence in *UTF-X* to the equivalent octet sequence in the native encoding (Latin-1 or EBCDIC). *$string* already encoded as octets does no harm. Returns true on success. On failure dies or, if the value of `FAIL_OK` is true, returns false. 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.

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

  Not affected by the encoding pragma.

  NOTE: this function is experimental and may change or be removed without notice.

* utf8::encode($string)

  Converts in-place the character sequence to the corresponding octet sequence in *UTF-X*. The UTF-8 flag is turned off. Returns nothing.

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

* utf8::decode($string)

  Attempts to convert in-place the octet sequence in *UTF-X* to the corresponding character sequence. 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.

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

  NOTE: this function is experimental and may change or be removed without notice.

* $flag = utf8::is_utf8(STRING)

  (Since Perl 5.8.1) Test whether STRING is 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 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`. Note that in the Perl 5.8.0 and 5.8.1 implementation the functions utf8::is_utf8, utf8::valid, utf8::encode, utf8::decode, utf8::upgrade, and utf8::downgrade are always available, without a `require utf8` statement-- this may change in future releases.

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

perluniintro, *encoding*, *perlrun*, *bytes*, *perlunicode*