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}"

# Convert a code point from the platform native character set to
# Unicode, and vice-versa.
$unicode = utf8::native_to_unicode(ord('A')); # returns 65 on both
                                              # ASCII and EBCDIC
                                              # platforms
$native = utf8::unicode_to_native(65);        # returns 65 on ASCII
                                              # platforms; 193 on
                                              # EBCDIC

$flag = utf8::is_utf8($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. The "no utf8" pragma tells Perl to switch back to treating the source text as literal bytes in the current lexical scope. (On EBCDIC platforms, technically it is allowing UTF-EBCDIC, and not UTF-8, but this distinction is academic, so in this document the term UTF-8 is used to mean both).

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.

See also the effects of the "-C" switch and its cousin, the "PERL_UNICODE" environment variable, in perlrun.

Enabling the "utf8" pragma has the following effect:

- Bytes in the source text that are not in the ASCII character set will be treated as being part of a literal UTF-8 sequence. This includes most literals such as identifier names, string constants, and constant regular expression patterns.

Note that if you have non-ASCII, non-UTF-8 bytes in your script (for example embedded Latin-1 in your string literals), "use utf8" will be unhappy. 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)"

  (Since Perl v5.8.0) Converts in-place the internal representation of the string from an octet sequence in the native encoding (Latin-1 or EBCDIC) to UTF-8. The logical character sequence itself is unchanged. If *$string* is already upgraded, then this is a no-op. Returns the number of octets necessary to represent the string as UTF-8.

  If your code needs to be compatible with versions of perl without "use feature ’unicode_strings’;", you can force Unicode semantics on a given string:

  ```
  # force unicode semantics for $string without the
  # "unicode_strings" feature
  utf8::upgrade($string);
  ```

  For example:

  ```
  # without explicit or implicit use feature ’unicode_strings’
  my $x = "\xDF";    # LATIN SMALL LETTER SHARP S
  ```

```
 $x =~ /ss/i;       # won't match
 my $y = uc($x);    # won't convert
 utf8::upgrade($x);
 $x =~ /ss/i;       # matches
 my $z = uc($x);    # converts to "SS"
```

Note that this function does not handle arbitrary encodings; use Encode instead.

- "$success = utf8::downgrade($string[, $fail_ok])"

  (Since Perl v5.8.0) Converts in-place the internal representation of the string from UTF-8 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-8 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.

  If your code expects an octet sequence this can be used to validate that you've received one:

  ```
   # throw an exception if not representable as octets
   utf8::downgrade($string)

   # or do your own error handling
   utf8::downgrade($string, 1) or die "string must be octets";
  ```

  Note that this function does not handle arbitrary encodings; use Encode instead.

- "utf8::encode($string)"

  (Since Perl v5.8.0) Converts in-place the character sequence to the corresponding octet sequence in Perl's extended UTF-8. That is, every (possibly wide) character gets replaced with a sequence of one or more characters that represent the individual UTF-8 bytes of the character. The UTF8 flag is turned off. Returns nothing.

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

```
                    # ASCII platforms) 0xc4 and 0x80.  On EBCDIC
                    # 1047, this would instead be 0x8C and 0x41.
```

Similar to:

```
  use Encode;
  $x = Encode::encode("utf8", $x);
```

Note that this function does not handle arbitrary encodings; use Encode instead.

- "$success = utf8::decode($string)"

  (Since Perl v5.8.0) Attempts to convert in-place the octet sequence encoded in Perl's extended UTF-8 to the corresponding character sequence. That is, it replaces each sequence of characters in the string whose ords represent a valid (extended) UTF-8 byte sequence, with the corresponding single character. The UTF-8 flag is turned on only if the source string contains multiple-byte UTF-8 characters. If *$string* is invalid as extended UTF-8, returns false; otherwise returns true.

```
 my $x = "\xc4\x80"; # $x contains two characters, with ords
                    # 0xc4 and 0x80
 utf8::decode($x);   # On ASCII platforms, $x contains one char,
                    # with ord 0x100.   Since these bytes aren't
                    # legal UTF-EBCDIC, on EBCDIC platforms, $x is
                    # unchanged and the function returns FALSE.
 my $y = "\xc3\x83\xc2\xab"; This has been encoded twice; this
                    # example is only for ASCII platforms
 utf8::decode($y);   # Converts $y to \xc3\xab, returns TRUE;
 utf8::decode($y);   # Further converts to \xeb, returns TRUE;
 utf8::decode($y);   # Returns FALSE, leaves $y unchanged
```

  Note that this function does not handle arbitrary encodings; use Encode instead.

- "$unicode = utf8::native_to_unicode($code_point)"

  (Since Perl v5.8.0) This takes an unsigned integer (which represents the ordinal number of a character (or a code point) on the platform the program is being run on) and returns its Unicode equivalent value. Since ASCII platforms natively use the Unicode code points, this function returns its input on them. On EBCDIC platforms it converts from EBCDIC to Unicode.

  A meaningless value will currently be returned if the input is not an unsigned integer.

Since Perl v5.22.0, calls to this function are optimized out on ASCII platforms, so there is no performance hit in using it there.

- "$native = utf8::unicode_to_native($code_point)"

  (Since Perl v5.8.0) This is the inverse of "utf8::native_to_unicode()", converting the other direction. Again, on ASCII platforms, this returns its input, but on EBCDIC platforms it will find the native platform code point, given any Unicode one.

  A meaningless value will currently be returned if the input is not an unsigned integer.

  Since Perl v5.22.0, calls to this function are optimized out on ASCII platforms, so there is no performance hit in using it there.

- "$flag = utf8::is_utf8($string)"

  (Since Perl 5.8.1) Test whether *$string* is marked internally as encoded in UTF-8. Functionally the same as "Encode::is_utf8($string)".

  Typically only necessary for debugging and testing, if you need to dump the internals of an SV, Devel::Peek's **Dump()** provides more detail in a compact form.

  If you still think you need this outside of debugging, testing or dealing with filenames, you should probably read perlunitut and "What is "the UTF8 flag"?" in perlunifaq.

  Don't use this flag as a marker to distinguish character and binary data: that should be decided for each variable when you write your code.

  To force unicode semantics in code portable to perl 5.8 and 5.10, call "utf8::upgrade($string)" unconditionally.

- "$flag = utf8::valid($string)"

  [INTERNAL] Test whether *$string* is in a consistent state regarding UTF-8. Will return true if it is well-formed Perl extended UTF-8 and has the UTF-8 flag on **or** if *$string* is held as bytes (both these states are 'consistent'). The main reason for this routine is to allow Perl's test suite to check that operations have left strings in a consistent state.

"utf8::encode" is like "utf8::upgrade", but the UTF8 flag is cleared. See perlunicode, and the C API functions "sv_utf8_upgrade", ""sv_utf8_downgrade" in perlapi", ""sv_utf8_encode" in perlapi", and ""sv_utf8_decode" in perlapi", 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

Some filesystems may not support UTF-8 file names, or they may be supported incompatibly with Perl. Therefore UTF-8 names that are visible to the filesystem, such as module names may not work.

SEE ALSO

perlunitut, perluniintro, perlrun, bytes, perlunicode