NAME

bytes - Perl pragma to expose the individual bytes of characters

NOTICE

Because the bytes pragma breaks encapsulation (i.e. it exposes the innards of how the perl executable currently happens to store a string), the byte values that result are in an unspecified encoding.

Use of this module for anything other than debugging purposes is strongly discouraged. If you feel that the functions here within might be useful for your application, this possibly indicates a mismatch between your mental model of Perl Unicode and the current reality. In that case, you may wish to read some of the perl Unicode documentation: *perluniintro*, *perlunitut*, *perlunifaq* and *perlunicode*.

SYNOPSIS

```
use bytes;
... chr(...);       # or bytes::chr
... index(...);     # or bytes::index
... length(...);    # or bytes::length
... ord(...);       # or bytes::ord
... rindex(...);    # or bytes::rindex
... substr(...);    # or bytes::substr
no bytes;
```

DESCRIPTION

Perl's characters are stored internally as sequences of one or more bytes. This pragma allows for the examination of the individual bytes that together comprise a character.

Originally the pragma was designed for the loftier goal of helping incorporate Unicode into Perl, but the approach that used it was found to be defective, and the one remaining legitimate use is for debugging when you need to non-destructively examine characters' individual bytes. Just insert this pragma temporarily, and remove it after the debugging is finished.

The original usage can be accomplished by explicit (rather than this pragma's implict) encoding using the *Encode* module:

```
use Encode qw/encode/;

my $utf8_byte_string   = encode "UTF8",   $string;
my $latin1_byte_string = encode "Latin1", $string;
```

Or, if performance is needed and you are only interested in the UTF-8 representation:

```
use utf8;

utf8::encode(my $utf8_byte_string = $string);
```

`no bytes` can be used to reverse the effect of `use bytes` within the current lexical scope.

As an example, when Perl sees `$x = chr(400)`, it encodes the character in UTF-8 and stores it in `$x`. Then it is marked as character data, so, for instance, `length $x` returns 1. However, in the scope of the `bytes` pragma, `$x` is treated as a series of bytes - the bytes that make up the UTF8 encoding - and `length $x` returns 2:

```
$x = chr(400);
print "Length is ", length $x, "\n";     # "Length is 1"
```

```
printf "Contents are %vd\n", $x;         # "Contents are 400"
{
    use bytes; # or "require bytes; bytes::length()"
    print "Length is ", length $x, "\n"; # "Length is 2"
    printf "Contents are %vd\n", $x;     # "Contents are 198.144 (on
                                         # ASCII platforms)"
}
```

`chr()`, `ord()`, `substr()`, `index()` and `rindex()` behave similarly.

For more on the implications, see *perluniintro* and *perlunicode*.

`bytes::length()` is admittedly handy if you need to know the **byte length** of a Perl scalar. But a more modern way is:

```
use Encode 'encode';
length(encode('UTF-8', $scalar))
```

LIMITATIONS

`bytes::substr()` does not work as an *lvalue()*.

SEE ALSO

perluniintro, *perlunicode*, *utf8*, *Encode*