NAME

CORE - Namespace for Perl's core routines

SYNOPSIS

```
BEGIN {
*CORE::GLOBAL::hex = sub { 1; };
}

print hex("0x50"),"\n"; # prints 1
print CORE::hex("0x50"),"\n"; # prints 80
CORE::say "yes"; # prints yes

BEGIN { *shove = \&CORE::push; }
shove @array, 1,2,3; # pushes on to @array
```

DESCRIPTION

The `CORE` namespace gives access to the original built-in functions of Perl. The `CORE` package is built into Perl, and therefore you do not need to use or require a hypothetical "CORE" module prior to accessing routines in this namespace.

A list of the built-in functions in Perl can be found in perlfunc.

For all Perl keywords, a `CORE::` prefix will force the built-in function to be used, even if it has been overridden or would normally require the feature pragma. Despite appearances, this has nothing to do with the CORE package, but is part of Perl's syntax.

For many Perl functions, the CORE package contains real subroutines. This feature is new in Perl 5.16. You can take references to these and make aliases. However, some can only be called as barewords; i.e., you cannot use ampersand syntax (`&foo`) or call them through references. See the `shove` example above. These subroutines exist for all keywords except the following:

`__DATA__`, `__END__`, `and`, `cmp`, `default`, `do`, `dump`, `else`, `elsif`, `eq`, `eval`, `for`, `foreach`, `format`, `ge`, `given`, `goto`, `grep`, `gt`, `if`, `last`, `le`, `local`, `lt`, `m`, `map`, `my`, `ne`, `next`, `no`, `or`, `our`, `package`, `print`, `printf`, `q`, `qq`, `qr`, `qw`, `qx`, `redo`, `require`, `return`, `s`, `say`, `sort`, `state`, `sub`, `tr`, `unless`, `until`, `use`, `when`, `while`, `x`, `xor`, `y`

Calling with ampersand syntax and through references does not work for the following functions, as they have special syntax that cannot always be translated into a simple list (e.g., `eof` vs `eof()`):

`chdir`, `chomp`, `chop`, `defined`, `delete`, `each`, `eof`, `exec`, `exists`, `keys`, `lstat`, `pop`, `push`, `shift`, `splice`, `split`, `stat`, `system`, `truncate`, `unlink`, `unshift`, `values`

OVERRIDING CORE FUNCTIONS

To override a Perl built-in routine with your own version, you need to import it at compile-time. This can be conveniently achieved with the `subs` pragma. This will affect only the package in which you've imported the said subroutine:

```
use subs 'chdir';
sub chdir { ... }
chdir $somewhere;
```

To override a built-in globally (that is, in all namespaces), you need to import your function into the `CORE::GLOBAL` pseudo-namespace at compile time:

```
BEGIN {
*CORE::GLOBAL::hex = sub {
# ... your code here
};
}
```

The new routine will be called whenever a built-in function is called without a qualifying package:

```
print hex("0x50"),"\n"; # prints 1
```

In both cases, if you want access to the original, unaltered routine, use the `CORE::` prefix:

```
print CORE::hex("0x50"),"\n"; # prints 80
```

AUTHOR

This documentation provided by Tels <nospam-abuse@bloodgate.com> 2007.

SEE ALSO

perlsub(1), perlfunc.