NAME

feature - Perl pragma to enable new features

SYNOPSIS

```
use feature qw(say switch);
given ($foo) {
    when (1)          { say "\$foo == 1" }
    when ([2,3])      { say "\$foo == 2 || \$foo == 3" }
    when (/^a[bc]d$/) { say "\$foo eq 'abd' || \$foo eq 'acd'" }
    when ($_ > 100)   { say "\$foo > 100" }
    default           { say "None of the above" }
}

use feature ':5.10'; # loads all features available in perl 5.10

use v5.10;           # implicitly loads :5.10 feature bundle
```

DESCRIPTION

It is usually impossible to add new syntax to Perl without breaking some existing programs. This pragma provides a way to minimize that risk. New syntactic constructs, or new semantic meanings to older constructs, can be enabled by `use feature 'foo'`, and will be parsed only when the appropriate feature pragma is in scope. (Nevertheless, the `CORE::` prefix provides access to all Perl keywords, regardless of this pragma.)

Lexical effect

Like other pragmas (`use strict`, for example), features have a lexical effect. `use feature qw(foo)` will only make the feature "foo" available from that point to the end of the enclosing block.

```
{
    use feature 'say';
    say "say is available here";
}
print "But not here.\n";
```

no feature

Features can also be turned off by using `no feature "foo"`. This too has lexical effect.

```
use feature 'say';
say "say is available here";
{
    no feature 'say';
    print "But not here.\n";
}
say "Yet it is here.";
```

`no feature` with no features specified will reset to the default group. To disable *all* features (an unusual request!) use `no feature ':all'`.

AVAILABLE FEATURES

The 'say' feature

`use feature 'say'` tells the compiler to enable the Perl 6 style `say` function.

See *"say" in perlfunc* for details.

This feature is available starting with Perl 5.10.

The 'state' feature

`use feature 'state'` tells the compiler to enable `state` variables.

See *"Persistent Private Variables" in perlsub* for details.

This feature is available starting with Perl 5.10.

The 'switch' feature

`use feature 'switch'` tells the compiler to enable the Perl 6 given/when construct.

See *"Switch Statements" in perlsyn* for details.

This feature is available starting with Perl 5.10.

The 'unicode_strings' feature

`use feature 'unicode_strings'` tells the compiler to use Unicode semantics in all string operations executed within its scope (unless they are also within the scope of either `use locale` or `use bytes`). The same applies to all regular expressions compiled within the scope, even if executed outside it. It does not change the internal representation of strings, but only how they are interpreted.

`no feature 'unicode_strings'` tells the compiler to use the traditional Perl semantics wherein the native character set semantics is used unless it is clear to Perl that Unicode is desired. This can lead to some surprises when the behavior suddenly changes. (See *"The "Unicode Bug"" in perlunicode* for details.) For this reason, if you are potentially using Unicode in your program, the `use feature 'unicode_strings'` subpragma is **strongly** recommended.

This feature is available starting with Perl 5.12; was almost fully implemented in Perl 5.14; and extended in Perl 5.16 to cover `quotemeta`.

The 'unicode_eval' and 'evalbytes' features

Under the `unicode_eval` feature, Perl's `eval` function, when passed a string, will evaluate it as a string of characters, ignoring any `use utf8` declarations. `use utf8` exists to declare the encoding of the script, which only makes sense for a stream of bytes, not a string of characters. Source filters are forbidden, as they also really only make sense on strings of bytes. Any attempt to activate a source filter will result in an error.

The `evalbytes` feature enables the `evalbytes` keyword, which evaluates the argument passed to it as a string of bytes. It dies if the string contains any characters outside the 8-bit range. Source filters work within `evalbytes`: they apply to the contents of the string being evaluated.

Together, these two features are intended to replace the historical `eval` function, which has (at least) two bugs in it, that cannot easily be fixed without breaking existing programs:

- `eval` behaves differently depending on the internal encoding of the string, sometimes treating its argument as a string of bytes, and sometimes as a string of characters.
- Source filters activated within `eval` leak out into whichever *file* scope is currently being compiled. To give an example with the CPAN module *Semi::Semicolons*:

  ```
  BEGIN { eval "use Semi::Semicolons;  # not filtered here " }
  # filtered here!
  ```

  `evalbytes` fixes that to work the way one would expect:

  ```
  use feature "evalbytes";
  BEGIN { evalbytes "use Semi::Semicolons;  # filtered " }
  # not filtered
  ```

These two features are available starting with Perl 5.16.

The 'current_sub' feature

This provides the `__SUB__` token that returns a reference to the current subroutine or `undef` outside of a subroutine.

This feature is available starting with Perl 5.16.

The 'array_base' feature

This feature supports the legacy `$[` variable. See *"$[" in perlvar* and *arybase*. It is on by default but disabled under `use v5.16` (see *IMPLICIT LOADING*, below).

This feature is available under this name starting with Perl 5.16. In previous versions, it was simply on all the time, and this pragma knew nothing about it.

The 'fc' feature

`use feature 'fc'` tells the compiler to enable the `fc` function, which implements Unicode casefolding.

See *"fc" in perlfunc* for details.

This feature is available from Perl 5.16 onwards.

The 'lexical_subs' feature

WARNING: This feature is still experimental and the implementation may change in future versions of Perl. For this reason, Perl will warn when you use the feature, unless you have explicitly disabled the warning:

```
    no warnings "experimental::lexical_subs";
```

This enables declaration of subroutines via `my sub foo`, `state sub foo` and `our sub foo` syntax. See *"Lexical Subroutines" in perlsub* for details.

This feature is available from Perl 5.18 onwards.

FEATURE BUNDLES

It's possible to load multiple features together, using a *feature bundle*. The name of a feature bundle is prefixed with a colon, to distinguish it from an actual feature.

```
  use feature ":5.10";
```

The following feature bundles are available:

```
  bundle    features included
  --------- -----------------
  :default  array_base

  :5.10     say state switch array_base

  :5.12     say state switch unicode_strings array_base

  :5.14     say state switch unicode_strings array_base

  :5.16     say state switch unicode_strings
            unicode_eval evalbytes current_sub fc
```

```
:5.18     say state switch unicode_strings
          unicode_eval evalbytes current_sub fc
```

The `:default` bundle represents the feature set that is enabled before any `use feature` or `no feature` declaration.

Specifying sub-versions such as the `0` in `5.14.0` in feature bundles has no effect. Feature bundles are guaranteed to be the same for all sub-versions.

```
use feature ":5.14.0";    # same as ":5.14"
use feature ":5.14.1";    # same as ":5.14"
```

IMPLICIT LOADING

Instead of loading feature bundles by name, it is easier to let Perl do implicit loading of a feature bundle for you.

There are two ways to load the `feature` pragma implicitly:

- By using the `-E` switch on the Perl command-line instead of `-e`. That will enable the feature bundle for that version of Perl in the main compilation unit (that is, the one-liner that follows `-E`).
- By explicitly requiring a minimum Perl version number for your program, with the `use VERSION` construct. That is,

  ```
  use v5.10.0;
  ```

  will do an implicit

  ```
  no feature ':all';
  use feature ':5.10';
  ```

  and so on. Note how the trailing sub-version is automatically stripped from the version.

  But to avoid portability warnings (see *"use" in perlfunc*), you may prefer:

  ```
  use 5.010;
  ```

  with the same effect.

  If the required version is older than Perl 5.10, the ":default" feature bundle is automatically loaded instead.