NAME

Exporter::Tiny::Manual::Importing – importing from Exporter::Tiny–based modules

DESCRIPTION

For the purposes of this discussion we'll assume we have a module called `MyUtils` which exports functions called `frobnicate`, `red`, `blue`, and `green`. It has a tag set up called `:colours` which corresponds to `red`, `blue`, and `green`.

Many of these tricks may seem familiar from Sub::Exporter. That is intentional. Exporter::Tiny doesn't attempt to provide every feature of Sub::Exporter, but where it does it usually uses a fairly similar API.

Basic importing

It's easy to import a single function from a module:

```
use MyUtils "frobnicate";
```

Or a list of functions:

```
use MyUtils "red", "green";
```

Perl's `qw()` shorthand for a list of words is pretty useful:

```
use MyUtils qw( red green );
```

If the module defines tags, you can import them like this:

```
use MyUtils qw( :colours );
```

Or with a hyphen instead of a colon:

```
use MyUtils qw( -colours );
```

Hyphens are good because Perl will autoquote a bareword that follows them:

```
use MyUtils -colours;
```

And it's possible to mix function names and tags in the same list:

```
use MyUtils qw( frobnicate :colours );
```

Renaming imported functions

It's possible to rename a function you're importing:

```
use MyUtils "frobnicate" => { -as => "frob" };
```

Or you can apply a prefix and/or suffix. The following imports the function and calls it `my_frobinate_thing`.

```
use MyUtils "frobnicate" => { -prefix => "my_", -suffix => "_thing" };
```

You can apply a prefix/suffix to **all** functions you import by placing the hashref **first** in the import list. (This first hashref is referred to as the global options hash, and can do some special things.)

```
use MyUtils { prefix => "my_" }, "frobnicate";
```

Did you notice that we used `-prefix` and `-suffix` in the normal options hash, but `prefix` and `suffix` (no hyphen) in the global options hash? That's a common pattern with this module.

You can import the same function multiple times with different names:

```
use MyUtils
   "frobnicate" => { -as => "frob" },
   "frobnicate" => { -as => "frbnct" };
```

Tags can take the `-prefix` and `-suffix` options too. The following imports `colour_red`, `colour_green`, and `colour_blue`:

```
use MyUtils -colours => { -prefix => "colour_" };
```

You can also set `-as` to be a coderef to generate a function name. This imports functions called `RED`, `GREEN`, and `BLUE`:

```
use MyUtils -colours => { -as => sub { uc($_[0]) } };
```

Note that it doesn't make sense to use `-as` with a tag unless you're doing this coderef thing. Coderef `as` also works in the global options hash.

DO NOT WANT!

Sometimes you want to supply a list of functions you **don't** want to import. To do that, prefix the function with a bang. This imports everything except ''frobnicate'':

```
use MyUtils qw( -all !frobnicate );
```

You can add the bang prefix to tags too. This will import everything except the colours.

```
use MyUtils qw( -all !:colours );
```

Negated imports always ''win'', so the following will not import ''frobnicate'', no matter how many times you repeat it...

```
use MyUtils qw( !frobnicate frobnicate frobnicate frobnicate );
```

Importing by regexp

Here's how you could import all functions beginning with an ''f'':

```
use MyUtils qw( /^F/i );
```

Or import everything except functions beginning with a ''z'':

```
use MyUtils qw( -all !/^Z/i );
```

Note that regexps are always supplied as *strings* starting with `"/"`, and not as quoted regexp references (`qr/.../`).

Import functions into another package

Occasionally you need to import functions not into your own package, but into a different package. You can do that like this:

```
use MyUtils { into => "OtherPkg" }, "frobnicate";

OtherPkg::frobincate(...);
```

However, Import::Into will probably provide you with a better approach which doesn't just work with Exporter::Tiny, but **all** exporters.

Lexical subs

Often you want to make use of an exported function, but don't want it to ''pollute'' your namespace.

There is this Sub::Exporter::Lexical thing that was designed as a plugin for Sub::Exporter, but Exporter::Tiny's API is close enough that it will work. Do you remember that global options hash? Just use that to tell Exporter::Tiny to use an alternative sub installer.

```
{
   use Sub::Exporter::Lexical lexical_installer => { -as => "lex" };
   use MyUtils { installer => lex }, "frobnicate";

   frobnicate(...);  # ok
}

frobnicate(...);  # not ok
```

Another way to do lexical functions is to import a function into a scalar variable:

```
my $func;
use MyUtils "frobnicate" => { -as => \$func };

$func->(...);
```

You can even provide a hashref to put all imported functions into as part of that global options hash I

mentioned earlier.

```
my %funcs;
use MyUtils { into => \%funcs }, "frobnicate";

$funcs{frobnicate}->(...);
```

Unimporting

You can unimport the functions that MyUtils added to your namespace:

```
no MyUtils;
```

Or just specific ones:

```
no MyUtils qw(frobnicate);
```

If you renamed a function when you imported it, you should unimport by the new name:

```
use MyUtils frobnicate => { -as => "frob" };
...;
no MyUtils "frob";
```

Unimporting using tags and regexps should mostly do what you want.

DIAGNOSTICS

Overwriting existing sub '%s::%s' with sub '%s' exported by `%s`

A warning issued if Exporter::Tiny is asked to export a symbol which will result in an existing sub being overwritten. This warning can be suppressed using either of the following:

```
use MyUtils { replace => 1 }, "frobnicate";
use MyUtils "frobnicate" => { -replace => 1 };
```

Or can be upgraded to a fatal error:

```
use MyUtils { replace => "die" }, "frobnicate";
use MyUtils "frobnicate" => { -replace => "die" };
```

Refusing to overwrite existing sub '%s::%s' with sub '%s' exported by `%s`

The fatal version of the above warning.

Could not find sub '%s' exported by `%s`

You requested to import a sub which the package does not provide.

Cannot provide an -as option for tags

Because a tag may provide more than one function, it does not make sense to request a single name for it. Instead use `-prefix` or `-suffix`.

Passing options to unimport '%s' makes no sense

When you import a sub, it occasionally makes sense to pass some options for it. However, when unimporting, options do nothing, so this warning is issued.

SEE ALSO

Exporter::Shiny, Exporter::Tiny.

AUTHOR

Toby Inkster <tobyink@cpan.org>.

COPYRIGHT AND LICENCE

DISCLAIMER OF WARRANTIES