NAME

Exporter::Tiny::Manual::QuickStart – the quickest way to get up and running with Exporter::Tiny

SYNOPSIS

```
package MyUtils;

use Exporter::Shiny qw( frobnicate );

sub frobnicate {
   ...;   # your code here
}

1;
```

Now people can use your module like this:

```
use MyUtils "frobnicate";

frobnicate(42);
```

Or like this:

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

frob(42);
```

DESCRIPTION

See the synopsis. Yes, it's that simple.

Next steps

Default exports

Note that the module in the synopsis doesn't export anything by default. If people load `MyUtils` like this:

```
use MyUtils;
```

Then they haven't imported any functions. You can specify a default set of functions to be exported like this:

```
package MyUtils;

use Exporter::Shiny qw( frobnicate );

our @EXPORT = qw( frobnicate );

sub frobnicate { ... }

1;
```

Or, if you want to be a superstar rock god:

```
package MyUtils;

use Exporter::Shiny our @EXPORT = qw( frobnicate );

sub frobnicate { ... }

1;
```

Tags

You can provide tags for people to use:

```
package MyUtils;

use Exporter::Shiny qw( frobnicate red green blue );

our %EXPORT_TAGS = (
   utils   => [qw/ frobnicate /],
   colours => [qw/ red green blue /],
);

sub frobnicate { ... }
sub red        { ... }
sub green      { ... }
sub blue       { ... }

1;
```

And people can now import your functions like this:

```
use MyUtils ":colours";
```

Or this:

```
use MyUtils "-colours";
```

Or take advantage of the fact that Perl magically quotes barewords preceded by a hyphen:

```
use MyUtils -colours;
```

Two tags are automatically defined for you: `-default` (which is just the same as `@EXPORT`) and `-all` (which is the union of `@EXPORT` and `@EXPORT_OK`). If you don't like them, then you can override them:

```
our %EXPORT_TAGS = (
   default => \@some_other_stuff,
   all     => \@more_stuff,
);
```

Generators

Exporting normally just works by copying a sub from your package into your caller's package. But sometimes it's useful instead to generate a *custom* sub to insert into your caller's package. This is pretty easy to do.

```
package MyUtils;

use Exporter::Shiny qw( frobnicate );

sub _generate_frobnicate {
   my $me     = shift;
   my $caller = caller;
   my ($name, $args) = @_;

   return sub {
       ...;  # your code here
   };
}

1;
```

The parameter `$me` here is a string containing the package name which is being imported from; `$caller` is the destination package; `$name` is the name of the sub (in this case "frobnicate"); and `$args` is a hashref of custom arguments for this function.

```
# The hashref { foo => 42 } is $args above.
#
use MyUtils "frobnicate" => { foo => 42 };
```

Avoiding Exporter::Shiny

Exporter::Shiny is a tiny shim around Exporter::Tiny. It should mostly do what you want, but you may sometimes prefer to use Exporter::Tiny directly.

The example in the synopsis could have been written as:

```
package MyUtils;

use parent "Exporter::Tiny";
our @EXPORT_OK = qw( frobnicate );

sub frobnicate {
   ...;   # your code here
}

1;
```

What Exporter::Shiny does is mostly just to set `@EXPORT_OK` for you and set up inheritance from the base class (Exporter::Tiny).

Exporter::Shiny also sets `$INC{'MyUtils.pm}` for you, which in usually makes little difference, but is useful in some edge cases.

SEE ALSO

Exporter::Shiny, Exporter::Tiny.

For more advanced information, see Exporter::Tiny::Manual::Exporting.

AUTHOR

Toby Inkster <tobyink@cpan.org>.

COPYRIGHT AND LICENCE

DISCLAIMER OF WARRANTIES