NAME

Exporter::Lite − lightweight exporting of functions and variables

SYNOPSIS

```
package Foo;
use Exporter::Lite;

our @EXPORT    = qw($This That);      # default exports
our @EXPORT_OK = qw(@Left %Right);    # optional exports
```

Then in code using the module:

```
use Foo;
# $This and &That are imported here
```

You have to explicitly ask for optional exports:

```
use Foo qw/ @Left %Right /;
```

DESCRIPTION

Exporter::Lite is an alternative to Exporter, intended to provide a lightweight subset of the most commonly-used functionality. It supports `import()`, `@EXPORT` and `@EXPORT_OK` and not a whole lot else.

Unlike Exporter, it is not necessary to inherit from Exporter::Lite; Ie you don't need to write:

```
@ISA = qw(Exporter::Lite);
```

Exporter::Lite simply exports its *import()* function into your namespace. This might be called a ''mix-in'' or a ''role''.

Setting up a module to export its variables and functions is simple:

```
package My::Module;
use Exporter::Lite;

our @EXPORT = qw($Foo bar);
```

Functions and variables listed in the `@EXPORT` package variable are automatically exported if you use the module and don't explicitly list any imports. Now, when you `use My::Module`, `$Foo` and `bar()` will show up.

Optional exports are listed in the `@EXPORT_OK` package variable:

```
package My::Module;
use Exporter::Lite;

our @EXPORT_OK = qw($Foo bar);
```

When My::Module is used, `$Foo` and `bar()` will *not* show up, unless you explicitly ask for them:

```
use My::Module qw($Foo bar);
```

Note that when you specify one or more functions or variables to import, then you must also explicitly list any of the default symbols you want to use. So if you have an exporting module:

```
package Games;
our @EXPORT    = qw/ pacman defender  /;
our @EXPORT_OK = qw/ galaga centipede /;
```

Then if you want to use both `pacman` and `galaga`, then you'd write:

```
use Games qw/ pacman galaga /;
```

Methods

Export::Lite has one public method, *import()*, which is called automatically when your modules is *use()*'d.

In normal usage you don't have to worry about this at all.

import

```
Some::Module->import;
Some::Module->import(@symbols);
```

Works just like `Exporter::import()` excepting it only honors `@Some::Module::EXPORT` and `@Some::Module::EXPORT_OK`.

The given `@symbols` are exported to the current package provided they are in `@Some::Module::EXPORT` or `@Some::Module::EXPORT_OK`. Otherwise an exception is thrown (ie. the program dies).

If `@symbols` is not given, everything in `@Some::Module::EXPORT` is exported.

DIAGNOSTICS

’“%s” is not exported by the `%s` module’

Attempted to import a symbol which is not in `@EXPORT` or `@EXPORT_OK`.

’Can\’t export symbol: `%s`’

Attempted to import a symbol of an unknown type (ie. the leading $@% salad wasn’t recognized).

SEE ALSO

Exporter is the grandaddy of all Exporter modules, and bundled with Perl itself, unlike the rest of the modules listed here.

Attribute::Exporter defines attributes which you use to mark which subs and variables you want to export, and how.

Exporter::Simple also uses attributes to control the export of functions and variables from your module.

Const::Exporter makes it easy to create a module that exports constants.

Constant::Exporter is another module that makes it easy to create modules that define and export constants.

Sub::Exporter is a “sophisticated exporter for custom-built routines”; it lets you provide generators that can be used to customise what gets imported when someone uses your module.

Exporter::Tiny provides the same features as Sub::Exporter, but relying only on core dependencies.

Exporter::Shiny is a shortcut for Exporter::Tiny that provides a more concise notation for providing optional exports.

Exporter::Declare provides syntactic sugar to make the export status of your functions part of their declaration. Kind of.

AppConfig::Exporter lets you export part of an AppConfig−based configuration.

Exporter::Lexical lets you export lexical subs from your module.

Constant::Export::Lazy lets you write a module that exports function-style constants, which are instantiated lazily.

Exporter::Auto will export everything from your module that it thinks is a public function (name doesn’t start with an underscore).

Class::Exporter lets you export class methods as regular subroutines.

Xporter is like Exporter, but with persistent defaults and auto-ISA.

REPOSITORY

<https://github.com/neilb/Exporter−Lite>

AUTHORS

Michael G Schwern <schwern AT pobox DOT com>

LICENSE