NAME

autouse - postpone load of modules until a function is used

SYNOPSIS

```
use autouse 'Carp' => qw(carp croak);
carp "this carp was predeclared and autoused ";
```

DESCRIPTION

If the module `Module` is already loaded, then the declaration

```
use autouse 'Module' => qw(func1 func2($;$));
```

is equivalent to

```
use Module qw(func1 func2);
```

if `Module` defines *func2()* with prototype `($;$)`, and *func1()* has no prototypes. (At least if `Module` uses `Exporter`'s `import`, otherwise it is a fatal error.)

If the module `Module` is not loaded yet, then the above declaration declares functions *func1()* and *func2()* in the current package. When these functions are called, they load the package `Module` if needed, and substitute themselves with the correct definitions.

WARNING

Using `autouse` will move important steps of your program's execution from compile time to runtime. This can

- Break the execution of your program if the module you `autoused` has some initialization which it expects to be done early.
- hide bugs in your code since important checks (like correctness of prototypes) is moved from compile time to runtime. In particular, if the prototype you specified on `autouse` line is wrong, you will not find it out until the corresponding function is executed. This will be very unfortunate for functions which are not always called (note that for such functions `autouse`ing gives biggest win, for a workaround see below).

To alleviate the second problem (partially) it is advised to write your scripts like this:

```
use Module;
use autouse Module => qw(carp($) croak(&$));
carp "this carp was predeclared and autoused ";
```

The first line ensures that the errors in your argument specification are found early. When you ship your application you should comment out the first line, since it makes the second one useless.

AUTHOR

Ilya Zakharevich (ilya@math.ohio-state.edu)

SEE ALSO

perl(1) .