Rocky Enterprise Linux 9.2 Manual Pages on command 'AutoLoader.3pm'

$ man AutoLoader.3pm

AutoLoader(3pm) Perl Programmers Reference Guide AutoLoader(3pm)

NAME

AutoLoader - load subroutines only on demand

SYNOPSIS

```
package Foo;
use AutoLoader 'AUTOLOAD';   # import the default AUTOLOAD subroutine
package Bar;
use AutoLoader;              # don't import AUTOLOAD, define our own
sub AUTOLOAD {
    ...
    $AutoLoader::AUTOLOAD = "...";
    goto &AutoLoader::AUTOLOAD;
}
```

DESCRIPTION

The AutoLoader module works with the AutoSplit module and the "__END__" token to defer the loading of some subroutines until they are used rather than loading them all at once.

To use AutoLoader, the author of a module has to place the definitions

of subroutines to be autoloaded after an "__END__" token. (See perldata.) The AutoSplit module can then be run manually to extract the definitions into individual files auto/funcname.al.

AutoLoader implements an AUTOLOAD subroutine. When an undefined subroutine in is called in a client module of AutoLoader, AutoLoader's AUTOLOAD subroutine attempts to locate the subroutine in a file with a name related to the location of the file from which the client module was read. As an example, if POSIX.pm is located in /usr/local/lib/perl5/POSIX.pm, AutoLoader will look for perl subroutines POSIX in /usr/local/lib/perl5/auto/POSIX/*.al, where the ".al" file has the same name as the subroutine, sans package. If such a file exists, AUTOLOAD will read and evaluate it, thus (presumably) defining the needed subroutine. AUTOLOAD will then "goto" the newly defined subroutine.

Once this process completes for a given function, it is defined, so future calls to the subroutine will bypass the AUTOLOAD mechanism.

Subroutine Stubs

In order for object method lookup and/or prototype checking to operate correctly even when methods have not yet been defined it is necessary to "forward declare" each subroutine (as in "sub NAME;"). See "SYNOPSIS" in perlsub. Such forward declaration creates "subroutine stubs", which are place holders with no code.

The AutoSplit and AutoLoader modules automate the creation of forward declarations. The AutoSplit module creates an 'index' file containing forward declarations of all the AutoSplit subroutines. When the AutoLoader module is 'use'd it loads these declarations into its callers package.

Because of this mechanism it is important that AutoLoader is always "use"d and not "require"d.

Using AutoLoader's AUTOLOAD Subroutine

In order to use AutoLoader's AUTOLOAD subroutine you must explicitly import it:

```
use AutoLoader 'AUTOLOAD';
```

Overriding AutoLoader's AUTOLOAD Subroutine

Some modules, mainly extensions, provide their own AUTOLOAD subroutines. They typically need to check for some special cases (such as constants) and then fallback to AutoLoader's AUTOLOAD for the rest. Such modules should not import AutoLoader's AUTOLOAD subroutine. Instead, they should define their own AUTOLOAD subroutines along these lines:

```
use AutoLoader;
use Carp;
sub AUTOLOAD {
    my $sub = $AUTOLOAD;
    (my $constname = $sub) =~ s/.*::/;
    my $val = constant($constname, @_ ? $_[0] : 0);
    if ($! != 0) {
        if ($! =~ /Invalid/ || $!{EINVAL}) {
            $AutoLoader::AUTOLOAD = $sub;
            goto &AutoLoader::AUTOLOAD;
        }
        else {
            croak "Your vendor has not defined constant $constname";
        }
    }
    *$sub = sub { $val }; # same as: eval "sub $sub { $val }";
    goto &$sub;
}
```

If any module's own AUTOLOAD subroutine has no need to fallback to the AutoLoader's AUTOLOAD subroutine (because it doesn't have any AutoSplit subroutines), then that module should not use AutoLoader at all.

Package Lexicals

Package lexicals declared with "my" in the main block of a package using AutoLoader will not be visible to auto-loaded subroutines, due to the fact that the given scope ends at the "__END__" marker. A module using such variables as package globals will not work properly under

the AutoLoader.

The "vars" pragma (see "vars" in perlmod) may be used in such situations as an alternative to explicitly qualifying all globals with the package namespace. Variables pre-declared with this pragma will be visible to any autoloaded routines (but will not be invisible outside the package, unfortunately).

Not Using AutoLoader

You can stop using AutoLoader by simply

```
no AutoLoader;
```

AutoLoader vs. SelfLoader

The AutoLoader is similar in purpose to SelfLoader: both delay the loading of subroutines.

SelfLoader uses the "__DATA__" marker rather than "__END__". While this avoids the use of a hierarchy of disk files and the associated open/close for each routine loaded, SelfLoader suffers a startup speed disadvantage in the one-time parsing of the lines after "__DATA__", after which routines are cached. SelfLoader can also handle multiple packages in a file.

AutoLoader only reads code as it is requested, and in many cases should be faster, but requires a mechanism like AutoSplit be used to create the individual files. ExtUtils::MakeMaker will invoke AutoSplit automatically if AutoLoader is used in a module source file.

Forcing AutoLoader to Load a Function

Sometimes, it can be necessary or useful to make sure that a certain function is fully loaded by AutoLoader. This is the case, for example, when you need to wrap a function to inject debugging code. It is also helpful to force early loading of code before forking to make use of copy-on-write as much as possible.

Starting with AutoLoader 5.73, you can call the "AutoLoader::autoload_sub" function with the fully-qualified name of the function to load from its .al file. The behaviour is exactly the same as if you called the function, triggering the regular "AUTOLOAD" mechanism, but it does not actually execute the autoloaded function.

CAVEATS

AutoLoaders prior to Perl 5.002 had a slightly different interface.

Any old modules which use AutoLoader should be changed to the new calling style. Typically this just means changing a require to a use, adding the explicit 'AUTOLOAD' import if needed, and removing AutoLoader from @ISA.

On systems with restrictions on file name length, the file corresponding to a subroutine may have a shorter name that the routine itself. This can lead to conflicting file names. The AutoSplit package warns of these potential conflicts when used to split a module.

AutoLoader may fail to find the autosplit files (or even find the wrong ones) in cases where @INC contains relative paths, and the program does "chdir".

SEE ALSO

SelfLoader - an autoloader that doesn't use external files.

AUTHOR

"AutoLoader" is maintained by the perl5-porters. Please direct any questions to the canonical mailing list. Anything that is applicable to the CPAN release can be sent to its maintainer, though.

Author and Maintainer: The Perl5-Porters <perl5-porters@perl.org>

Maintainer of the CPAN release: Steffen Mueller <smueller@cpan.org>

COPYRIGHT AND LICENSE

a) the GNU General Public License as published by the Free
Software Foundation; either version 1, or (at your option) any
later version, or
b) the "Artistic License" which comes with this Kit.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See either
the GNU General Public License or the Artistic License for more details.
You should have received a copy of the Artistic License with this
Kit, in the file named "Artistic". If not, I'll be glad to provide one.
You should also have received a copy of the GNU General Public License
along with this program in the file named "Copying". If not, write to the
Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston,
MA 02110-1301, USA or visit their web page on the internet at
http://www.gnu.org/copyleft/gpl.html.
For those of you that choose to use the GNU General Public License,
my interpretation of the GNU General Public License is that no Perl
script falls under the terms of the GPL unless you explicitly put
said script under the terms of the GPL yourself. Furthermore, any
object code linked with perl does not automatically fall under the
terms of the GPL, provided such object code only adds definitions
of subroutines and variables, and does not otherwise impair the
resulting interpreter from executing any standard Perl script. I
consider linking in C subroutines in this manner to be the moral
equivalent of defining subroutines in the Perl language itself. You
may sell such an object file as proprietary provided that you provide
or offer to provide the Perl source, as specified by the GNU General
Public License. (This is merely an alternate way of specifying input
to the program.) You may also sell a binary produced by the dumping of
a running Perl script that belongs to you, provided that you provide or
offer to provide the Perl source as specified by the GPL. (The
fact that a Perl interpreter and your code are in the same binary file
is, in this case, a form of mere aggregation.) This is my interpretation

of the GPL. If you still have concerns or difficulties understanding my intent, feel free to contact me. Of course, the Artistic License spells all this out for your protection, so you may prefer to use that.