NAME

Module::Load::Conditional - Looking up module information / loading at runtime

SYNOPSIS

```
use Module::Load::Conditional qw[can_load check_install requires];


my $use_list = {
CPANPLUS => 0.05,
LWP => 5.60,
'Test::More' => undef,
};

print can_load( modules => $use_list )
? 'all modules loaded successfully'
: 'failed to load required modules';


my $rv = check_install( module => 'LWP', version => 5.60 )
or print 'LWP is not installed!';

print 'LWP up to date' if $rv->{uptodate};
print "LWP version is $rv->{version}\n";
print "LWP is installed as file $rv->{file}\n";


print "LWP requires the following modules to be installed:\n";
print join "\n", requires('LWP');

### allow M::L::C to peek in your %INC rather than just
### scanning @INC
$Module::Load::Conditional::CHECK_INC_HASH = 1;

### reset the 'can_load' cache
undef $Module::Load::Conditional::CACHE;

### don't have Module::Load::Conditional issue warnings --
### default is '1'
$Module::Load::Conditional::VERBOSE = 0;

### The last error that happened during a call to 'can_load'
my $err = $Module::Load::Conditional::ERROR;
```

DESCRIPTION

Module::Load::Conditional provides simple ways to query and possibly load any of the modules you have installed on your system during runtime.

It is able to load multiple modules at once or none at all if one of them was not able to load. It also takes care of any error checking and so forth.

Methods

$href = check_install(module => NAME [, version => VERSION, verbose => BOOL]);

`check_install` allows you to verify if a certain module is installed or not. You may call it with the following arguments:

module
: The name of the module you wish to verify — this is a required key

version
: The version this module needs to be — this is optional

verbose
: Whether or not to be verbose about what it is doing — it will default to `$Module::Load::Conditional::VERBOSE`

It will return undef if it was not able to find where the module was installed, or a hash reference with the following keys if it was able to find the file:

file Full path to the file that contains the module

dir Directory, or more exact the `@INC` entry, where the module was loaded from.

version
: The version number of the installed module - this will be `undef` if the module had no (or unparsable) version number, or if the variable `$Module::Load::Conditional::FIND_VERSION` was set to true. (See the `GLOBAL VARIABLES` section below for details)

uptodate
: A boolean value indicating whether or not the module was found to be at least the version you specified. If you did not specify a version, uptodate will always be true if the module was found. If no parsable version was found in the module, uptodate will also be true, since `check_install` had no way to verify clearly.

  See also `$Module::Load::Conditional::DEPRECATED`, which affects the outcome of this value.

`$bool` = can_load(modules => { NAME => VERSION [,NAME => VERSION] }, [verbose => BOOL, nocache => BOOL, autoload => BOOL])

`can_load` will take a list of modules, optionally with version numbers and determine if it is able to load them. If it can load *ALL* of them, it will. If one or more are unloadable, none will be loaded.

This is particularly useful if you have More Than One Way (tm) to solve a problem in a program, and only wish to continue down a path if all modules could be loaded, and not load them if they couldn't.

This function uses the `load` function or the `autoload_remote` function from Module::Load under the hood.

`can_load` takes the following arguments:

modules
: This is a hashref of module/version pairs. The version indicates the minimum version to load. If no version is provided, any version is assumed to be good enough.

verbose
: This controls whether warnings should be printed if a module failed to load. The default is to use the value of `$Module::Load::Conditional::VERBOSE`.

nocache
: `can_load` keeps its results in a cache, so it will not load the same module twice, nor will it attempt to load a module that has already failed to load before. By default, `can_load` will check its cache, but you can override that by setting `nocache` to true.

autoload
: This controls whether imports the functions of a loaded modules to the caller package. The default is no importing any functions.

See the `autoload` function and the `autoload_remote` function from Module::Load for details.

`@list = requires( MODULE );`

`requires` can tell you what other modules a particular module requires. This is particularly useful when you're intending to write a module for public release and are listing its prerequisites.

`requires` takes but one argument: the name of a module. It will then first check if it can actually load this module, and return undef if it can't. Otherwise, it will return a list of modules and pragmas that would have been loaded on the module's behalf.

Note: The list `require` returns has originated from your current perl and your current install.

Global Variables

The behaviour of Module::Load::Conditional can be altered by changing the following global variables:

`$Module::Load::Conditional::VERBOSE`

This controls whether Module::Load::Conditional will issue warnings and explanations as to why certain things may have failed. If you set it to 0, Module::Load::Conditional will not output any warnings. The default is 0;

`$Module::Load::Conditional::FIND_VERSION`

This controls whether Module::Load::Conditional will try to parse (and eval) the version from the module you're trying to load.

If you don't wish to do this, set this variable to `false`. Understand then that version comparisons are not possible, and Module::Load::Conditional can not tell you what module version you have installed. This may be desirable from a security or performance point of view. Note that `$FIND_VERSION` code runs safely under `taint mode`.

The default is 1;

`$Module::Load::Conditional::CHECK_INC_HASH`

This controls whether `Module::Load::Conditional` checks your `%INC` hash to see if a module is available. By default, only `@INC` is scanned to see if a module is physically on your filesystem, or available via an `@INC-hook`. Setting this variable to `true` will trust any entries in `%INC` and return them for you.

The default is 0;

`$Module::Load::Conditional::CACHE`

This holds the cache of the `can_load` function. If you explicitly want to remove the current cache, you can set this variable to `undef`

`$Module::Load::Conditional::ERROR`

This holds a string of the last error that happened during a call to `can_load`. It is useful to inspect this when `can_load` returns `undef`.

`$Module::Load::Conditional::DEPRECATED`

This controls whether `Module::Load::Conditional` checks if a dual-life core module has been deprecated. If this is set to true `check_install` will return false to `uptodate`, if a dual-life module is found to be loaded from `$Config{privlibexp}`

The default is 0;

See Also

`Module::Load`

BUG REPORTS

Please report bugs or other issues to <bug-module-load-conditional@rt.cpan.org>.

AUTHOR

This module by Jos Boumans <kane@cpan.org>.

COPYRIGHT