NAME

ExtUtils::Depends − Easily build XS extensions that depend on XS extensions

SYNOPSIS

```
use ExtUtils::Depends;
$package = new ExtUtils::Depends ('pkg::name', 'base::package')
# set the flags and libraries to compile and link the module
$package->set_inc("-I/opt/blahblah");
$package->set_libs("-lmylib");
# add a .c and an .xs file to compile
$package->add_c('code.c');
$package->add_xs('module-code.xs');
# add the typemaps to use
$package->add_typemaps("typemap");
# install some extra data files and headers
$package->install (qw/foo.h data.txt/);
# save the info
$package->save_config('Files.pm');

WriteMakefile(
        'NAME' => 'Mymodule',
        $package->get_makefile_vars()
);
```

DESCRIPTION

This module tries to make it easy to build Perl extensions that use functions and typemaps provided by other perl extensions. This means that a perl extension is treated like a shared library that provides also a C and an XS interface besides the perl one.

This works as long as the base extension is loaded with the RTLD_GLOBAL flag (usually done with a

```
sub dl_load_flags {0x01}
```

in the main .pm file) if you need to use functions defined in the module.

The basic scheme of operation is to collect information about a module in the instance, and then store that data in the Perl library where it may be retrieved later. The object can also reformat this information into the data structures required by ExtUtils::MakeMaker's WriteMakefile function.

For information on how to make your module fit into this scheme, see ''hashref = ExtUtils::Depends::load (name)''.

When creating a new Depends object, you give it a name, which is the name of the module you are building. You can also specify the names of modules on which this module depends. These dependencies will be loaded automatically, and their typemaps, header files, etc merged with your new object's stuff. When you store the data for your object, the list of dependencies are stored with it, so that another module depending on your needn't know on exactly which modules yours depends.

For example:

```
Gtk2 depends on Glib

Gnome2::Canvas depends on Gtk2

ExtUtils::Depends->new ('Gnome2::Canvas', 'Gtk2');
   this command automatically brings in all the stuff needed
   for Glib, since Gtk2 depends on it.
```

When the configuration information is saved, it also includes a class method called `Inline`, inheritable by your module. This allows you in your module to simply say at the top:

```
  package Mymod;
  use parent 'Mymod::Install::Files'; # to inherit 'Inline' method
```

And users of `Mymod` who want to write inline code (using Inline) will simply be able to write:

```
  use Inline with => 'Mymod';
```

And all the necessary header files, defines, and libraries will be added for them.

The `Mymod::Install::Files` will also implement a `deps` method, which will return a list of any modules that `Mymod` depends on − you will not normally need to use this:

```
  require Mymod::Install::Files;
  @deps = Mymod::Install::Files->deps;
```

METHODS

`$object` = ExtUtils::Depends->new($name, `@deps`)

Create a new depends object named *`$name`*. Any modules listed in *`@deps`* (which may be empty) are added as dependencies and their dependency information is loaded. An exception is raised if any dependency information cannot be loaded.

`$depends`->add_deps (@deps)

Add modules listed in *`@deps`* as dependencies.

(hashes) = `$depends`->get_deps

Fetch information on the dependencies of *`$depends`* as a hash of hashes, which are dependency information indexed by module name. See `load`.

`$depends`->set_inc (@newinc)

Add strings to the includes or cflags variables.

`$depends`->set_libs (@newlibs)

Add strings to the libs (linker flags) variable.

`$depends`->add_pm (%pm_files)

Add files to the hash to be passed through ExtUtils::WriteMakefile's PM key.

`$depends`->add_xs (@xs_files)

Add xs files to be compiled.

`$depends`->add_c (@c_files)

Add C files to be compiled.

`$depends`->add_typemaps (@typemaps)

Add typemap files to be used and installed.

`$depends`->add_headers (list)

No-op, for backward compatibility.

`$depends`->install (@files)

Install *`@files`* to the data directory for *`$depends`*.

This actually works by adding them to the hash of pm files that gets passed through WriteMakefile's PM key.

`$depends`->save_config ($filename)

Save the important information from *`$depends`* to *`$filename`*, and set it up to be installed as *name*::Install::Files.

Note: the actual value of *`$filename`* is unimportant so long as it doesn't clash with any other local files. It will be installed as *name*::Install::Files.

hash = `$depends`->get_makefile_vars

Return the information in *`$depends`* in a format digestible by WriteMakefile.

This sets at least the following keys:

```
INC
LIBS
TYPEMAPS
PM
```

And these if there is data to fill them:

```
clean
OBJECT
XS
```

hashref = ExtUtils::Depends::load (name)

Load and return dependency information for *name*. Croaks if no such information can be found. The information is returned as an anonymous hash containing these keys:

instpath
: The absolute path to the data install directory for this module.

typemaps
: List of absolute pathnames for this module's typemap files.

inc CFLAGS string for this module.

libs LIBS string for this module.

deps
: List of modules on which this one depends. This key will not exist when loading files created by old versions of ExtUtils::Depends.

If you want to make module *name* support this, you must provide a module *name*::Install::Files, which on loading will implement the following class methods:

```
$hashref = name::Install::Files->Inline('C');
# hash to contain any necessary TYPEMAPS (array-ref), LIBS, INC
@deps = name::Install::Files->deps;
# any modules on which "name" depends
```

An easy way to achieve this is to use the method ''$depends->save_config ($filename)'', but your package may have different facilities already.

`$depends`->load_deps

Load *`$depends`* dependencies, by calling `load` on each dependency module. This is usually done for you, and should only be needed if you want to call `get_deps` after calling `add_deps` manually.

SUPPORT

Bugs/Feature Requests

Version 0.2 discards some of the more esoteric features provided by the older versions. As they were completely undocumented, and this module has yet to reach 1.0, this may not exactly be a bug.

This module is tightly coupled to the ExtUtils::MakeMaker architecture.

You can submit new bugs/feature requests by using one of two bug trackers (below).

CPAN Request Tracker
: You can submit bugs/feature requests via the web by going to <https://rt.cpan.org/Public/Bug/Report.html?Queue=ExtUtils-Depends> (requires PAUSE ID or Bitcard), or by sending an e-mail to ''bug-ExtUtils-Depends at rt.cpan.org''.

Gnome.org Bugzilla
: Report bugs/feature requests to the 'gnome-perl' product (requires login) <http://bugzilla.gnome.org/enter_bug.cgi?product=gnome-perl>

Patches that implement new features with test cases, and/or test cases that exercise existing bugs are always welcome.

The Gtk-Perl mailing list is at ‘‘gtk-perl-list at gnome dot org’’.

Source Code

The source code to ExtUtils::Depends is available at the Gnome.org Git repo (<https://git.gnome.org/browse/perl−ExtUtils−Depends/>). Create your own copy of the Git repo with:

```
git clone git://git.gnome.org/perl−ExtUtils−Depends (Git protocol)
git clone https://git.gnome.org/browse/perl−ExtUtils−Depends/ (HTTPS)
```

SEE ALSO

ExtUtils::MakeMaker.

AUTHOR

Paolo Molaro <lupus at debian dot org> wrote the original version for Gtk-Perl. muppet <scott at asofyet dot org> rewrote the innards for version 0.2, borrowing liberally from Paolo’s code.

MAINTAINER

The Gtk2 project, <http://gtk2−perl.sf.net>/‘‘gtk-perl-list at gnome dot org’’.

LICENSE