NAME

Module::Build::Compat - Compatibility with ExtUtils::MakeMaker

SYNOPSIS

```
# In a Build.PL :
use Module::Build;
my $build = Module::Build->new
( module_name => 'Foo::Bar',
license => 'perl',
create_makefile_pl => 'traditional' );
...
```

DESCRIPTION

Because ExtUtils::MakeMaker has been the standard way to distribute modules for a long time, many tools (CPAN.pm, or your system administrator) may expect to find a working *Makefile.PL* in every distribution they download from CPAN. If you want to throw them a bone, you can use Module::Build::Compat to automatically generate a *Makefile.PL* for you, in one of several different styles.

Module::Build::Compat also provides some code that helps out the *Makefile.PL* at runtime.

METHODS

create_makefile_pl($style, `$build`)

Creates a *Makefile.PL* in the current directory in one of several styles, based on the supplied Module::Build object `$build`. This is typically controlled by passing the desired style as the `create_makefile_pl` parameter to Module::Build `new()` method; the *Makefile.PL* will then be automatically created during the `distdir` action.

The currently supported styles are:

traditional

A *Makefile.PL* will be created in the "traditional" style, i.e. it will use ExtUtils::MakeMaker and won't rely on Module::Build at all. In order to create the *Makefile.PL*, we'll include the `requires` and `build_requires` dependencies as the `PREREQ_PM` parameter.

You don't want to use this style if during the `perl Build.PL` stage you ask the user questions, or do some auto-sensing about the user's environment, or if you subclass Module::Build to do some customization, because the vanilla *Makefile.PL* won't do any of that.

small

A small *Makefile.PL* will be created that passes all functionality through to the *Build.PL* script in the same directory. The user must already have Module::Build installed in order to use this, or else they'll get a module-not-found error.

passthrough (DEPRECATED)

This is just like the `small` option above, but if Module::Build is not already installed on the user's system, the script will offer to use `CPAN.pm` to download it and install it before continuing with the build.

This option has been deprecated and may be removed in a future version of Module::Build. Modern CPAN.pm and CPANPLUS will recognize the `configure_requires` metadata property and install Module::Build before running Build.PL if Module::Build is listed and Module::Build now adds itself to configure_requires by default.

Perl 5.10.1 includes `configure_requires` support. In the future, when `configure_requires` support is deemed sufficiently widespread, the `passthrough` style will be removed.

run_build_pl(args => @ARGV)
> This method runs the *Build.PL* script, passing it any arguments the user may have supplied to the `perl Makefile.PL` command. Because `ExtUtils::MakeMaker` and `Module::Build` accept different arguments, this method also performs some translation between the two.
>
> `run_build_pl()` accepts the following named parameters:
>
> args
>> The `args` parameter specifies the parameters that would usually appear on the command line of the `perl Makefile.PL` command - typically you'll just pass a reference to `@ARGV`.
>
> script
>> This is the filename of the script to run - it defaults to `Build.PL`.

write_makefile()
> This method writes a 'dummy' *Makefile* that will pass all commands through to the corresponding `Module::Build` actions.
>
> `write_makefile()` accepts the following named parameters:
>
> makefile
>> The name of the file to write - defaults to the string `Makefile`.

SCENARIOS
So, some common scenarios are:

1. Just include a *Build.PL* script (without a *Makefile.PL* script), and give installation directions in a *README* or *INSTALL* document explaining how to install the module. In particular, explain that the user must install `Module::Build` before installing your module.

   Note that if you do this, you may make things easier for yourself, but harder for people with older versions of CPAN or CPANPLUS on their system, because those tools generally only understand the *Makefile.PL*/ExtUtils::MakeMaker way of doing things.

2. Include a *Build.PL* script and a "traditional" *Makefile.PL*, created either manually or with `create_makefile_pl()`. Users won't ever have to install `Module::Build` if they use the *Makefile.PL*, but they won't get to take advantage of `Module::Build` extra features either.

   For good measure, of course, test both the *Makefile.PL* and the *Build.PL* before shipping.

3. Include a *Build.PL* script and a "pass-through" *Makefile.PL* built using `Module::Build::Compat` This will mean that people can continue to use the "old" installation commands, and they may never notice that it's actually doing something else behind the scenes. It will also mean that your installation process is compatible with older versions of tools like CPAN and CPANPLUS.

AUTHOR
Ken Williams <kwilliams@cpan.org>

COPYRIGHT
Copyright (c) 2001-2006 Ken Williams. All rights reserved.

This library is free software; you can redistribute it and/or modify it under the same terms as Perl itself.

SEE ALSO
Module::Build(3), ExtUtils::MakeMaker(3)