NAME

ExtUtils::CBuilder - Compile and link C code for Perl modules

SYNOPSIS

```
use ExtUtils::CBuilder;

my $b = ExtUtils::CBuilder->new(%options);
$obj_file = $b->compile(source => 'MyModule.c');
$lib_file = $b->link(objects => $obj_file);
```

DESCRIPTION

This module can build the C portions of Perl modules by invoking the appropriate compilers and linkers in a cross-platform manner. It was motivated by the `Module::Build` project, but may be useful for other purposes as well. However, it is *not* intended as a general cross-platform interface to all your C building needs. That would have been a much more ambitious goal!

METHODS

new

Returns a new `ExtUtils::CBuilder` object. A `config` parameter lets you override `Config.pm` settings for all operations performed by the object, as in the following example:

```
# Use a different compiler than Config.pm says
my $b = ExtUtils::CBuilder->new( config =>
                                 { ld => 'gcc' } );
```

A `quiet` parameter tells `CBuilder` to not print its `system()` commands before executing them:

```
# Be quieter than normal
my $b = ExtUtils::CBuilder->new( quiet => 1 );
```

have_compiler

Returns true if the current system has a working C compiler and linker, false otherwise. To determine this, we actually compile and link a sample C library. The sample will be compiled in the system tempdir or, if that fails for some reason, in the current directory.

have_cplusplus

Just like have_compiler but for C++ instead of C.

compile

Compiles a C source file and produces an object file. The name of the object file is returned. The source file is specified in a `source` parameter, which is required; the other parameters listed below are optional.

`object_file`

Specifies the name of the output file to create. Otherwise the `object_file()` method will be consulted, passing it the name of the `source` file.

`include_dirs`

Specifies any additional directories in which to search for header files. May be given as a string indicating a single directory, or as a list reference indicating multiple directories.

`extra_compiler_flags`

Specifies any additional arguments to pass to the compiler. Should be given as a list

reference containing the arguments individually, or if this is not possible, as a string containing all the arguments together.

C++

Specifies that the source file is a C++ source file and sets appropriate compiler flags

The operation of this method is also affected by the `archlibexp`, `cccdlflags`, `ccflags`, `optimize`, and `cc` entries in `Config.pm`.

link

Invokes the linker to produce a library file from object files. In scalar context, the name of the library file is returned. In list context, the library file and any temporary files created are returned. A required `objects` parameter contains the name of the object files to process, either in a string (for one object file) or list reference (for one or more files). The following parameters are optional:

lib_file

Specifies the name of the output library file to create. Otherwise the `lib_file()` method will be consulted, passing it the name of the first entry in `objects`.

module_name

Specifies the name of the Perl module that will be created by linking. On platforms that need to do prelinking (Win32, OS/2, etc.) this is a required parameter.

extra_linker_flags

Any additional flags you wish to pass to the linker.

On platforms where `need_prelink()` returns true, `prelink()` will be called automatically.

The operation of this method is also affected by the `lddlflags`, `shrpenv`, and `ld` entries in `Config.pm`.

link_executable

Invokes the linker to produce an executable file from object files. In scalar context, the name of the executable file is returned. In list context, the executable file and any temporary files created are returned. A required `objects` parameter contains the name of the object files to process, either in a string (for one object file) or list reference (for one or more files). The optional parameters are the same as `link` with exception for

exe_file

Specifies the name of the output executable file to create. Otherwise the `exe_file()` method will be consulted, passing it the name of the first entry in `objects`.

object_file

```
my $object_file = $b->object_file($source_file);
```

Converts the name of a C source file to the most natural name of an output object file to create from it. For instance, on Unix the source file *foo.c* would result in the object file *foo.o*.

lib_file

```
my $lib_file = $b->lib_file($object_file);
```

Converts the name of an object file to the most natural name of a output library file to create from it. For instance, on Mac OS X the object file *foo.o* would result in the library file *foo.bundle*.

exe_file

```
my $exe_file = $b->exe_file($object_file);
```

Converts the name of an object file to the most natural name of an executable file to create from it. For instance, on Mac OS X the object file *foo.o* would result in the executable file *foo*, and on Windows it would result in *foo.exe*.

prelink

On certain platforms like Win32, OS/2, VMS, and AIX, it is necessary to perform some actions before invoking the linker. The `ExtUtils::Mksymlists` module does this, writing files used by the linker during the creation of shared libraries for dynamic extensions. The names of any files written will be returned as a list.

Several parameters correspond to `ExtUtils::Mksymlists::Mksymlists()` options, as follows:

```
 Mksymlists()   prelink()          type
-------------|------------------|------------------
 NAME        |  dl_name         | string (required)
 DLBASE      |  dl_base         | string
 FILE        |  dl_file         | string
 DL_VARS     |  dl_vars         | array reference
 DL_FUNCS    |  dl_funcs        | hash reference
 FUNCLIST    |  dl_func_list    | array reference
 IMPORTS     |  dl_imports      | hash reference
 VERSION     |  dl_version      | string
```

Please see the documentation for `ExtUtils::Mksymlists` for the details of what these parameters do.

need_prelink

Returns true on platforms where `prelink()` should be called during linking, and false otherwise.

extra_link_args_after_prelink

Returns list of extra arguments to give to the link command; the arguments are the same as for prelink(), with addition of array reference to the results of prelink(); this reference is indexed by key `prelink_res`.

TO DO

Currently this has only been tested on Unix and doesn't contain any of the Windows-specific code from the `Module::Build` project. I'll do that next.

HISTORY

This module is an outgrowth of the `Module::Build` project, to which there have been many contributors. Notably, Randy W. Sims submitted lots of code to support 3 compilers on Windows and helped with various other platform-specific issues. Ilya Zakharevich has contributed fixes for OS/2; John E. Malmberg and Peter Prymmer have done likewise for VMS.

AUTHOR

Ken Williams, kwilliams@cpan.org

COPYRIGHT

SEE ALSO

perl(1), Module::Build(3)