NAME

ExtUtils::Liblist - determine libraries to use and how to use them

SYNOPSIS

```
require ExtUtils::Liblist;

$MM->ext($potential_libs, $verbose, $need_names);

# Usually you can get away with:
ExtUtils::Liblist->ext($potential_libs, $verbose, $need_names)
```

DESCRIPTION

This utility takes a list of libraries in the form `-llib1 -llib2 -llib3` and returns lines suitable for inclusion in an extension Makefile. Extra library paths may be included with the form `-L/another/path` this will affect the searches for all subsequent libraries.

It returns an array of four or five scalar values: EXTRALIBS, BSLOADLIBS, LDLOADLIBS, LD_RUN_PATH, and, optionally, a reference to the array of the filenames of actual libraries. Some of these don't mean anything unless on Unix. See the details about those platform specifics below. The list of the filenames is returned only if $need_names argument is true.

Dependent libraries can be linked in one of three ways:

* For static extensions

  by the ld command when the perl binary is linked with the extension library. See EXTRALIBS below.

* For dynamic extensions at build/link time

  by the ld command when the shared object is built/linked. See LDLOADLIBS below.

* For dynamic extensions at load time

  by the DynaLoader when the shared object is loaded. See BSLOADLIBS below.

EXTRALIBS

List of libraries that need to be linked with when linking a perl binary which includes this extension. Only those libraries that actually exist are included. These are written to a file and used when linking perl.

LDLOADLIBS and LD_RUN_PATH

List of those libraries which can or must be linked into the shared library when created using ld. These may be static or dynamic libraries. LD_RUN_PATH is a colon separated list of the directories in LDLOADLIBS. It is passed as an environment variable to the process that links the shared library.

BSLOADLIBS

List of those libraries that are needed but can be linked in dynamically at run time on this platform. SunOS/Solaris does not need this because ld records the information (from LDLOADLIBS) into the object file. This list is used to create a .bs (bootstrap) file.

PORTABILITY

This module deals with a lot of system dependencies and has quite a few architecture specific `ifs` in the code.

VMS implementation

The version of ext() which is executed under VMS differs from the Unix-OS/2 version in several respects:

- Input library and path specifications are accepted with or without the `-l` and `-L` prefixes used by Unix linkers. If neither prefix is present, a token is considered a directory to search if it is in fact a directory, and a library to search for otherwise. Authors who wish their extensions to be portable to Unix or OS/2 should use the Unix prefixes, since the Unix-OS/2 version of ext() requires them.

- Wherever possible, shareable images are preferred to object libraries, and object libraries to plain object files. In accordance with VMS naming conventions, ext() looks for files named *lib*shr and *lib*rtl; it also looks for *lib*lib and lib*lib* to accommodate Unix conventions used in some ported software.

- For each library that is found, an appropriate directive for a linker options file is generated. The return values are space-separated strings of these directives, rather than elements used on the linker command line.

- LDLOADLIBS contains both the libraries found based on `$potential_libs` and the CRTLs, if any, specified in Config.pm. EXTRALIBS contains just those libraries found based on `$potential_libs`. BSLOADLIBS and LD_RUN_PATH are always empty.

In addition, an attempt is made to recognize several common Unix library names, and filter them out or convert them to their VMS equivalents, as appropriate.

In general, the VMS version of ext() should properly handle input from extensions originally designed for a Unix or VMS environment. If you encounter problems, or discover cases where the search could be improved, please let us know.

Win32 implementation

The version of ext() which is executed under Win32 differs from the Unix-OS/2 version in several respects:

- If `$potential_libs` is empty, the return value will be empty. Otherwise, the libraries specified by `$Config{perllibs}` (see Config.pm) will be appended to the list of `$potential_libs`. The libraries will be searched for in the directories specified in `$potential_libs`, `$Config{libpth}`, and in `$Config{installarchlib}/CORE`. For each library that is found, a space-separated list of fully qualified library pathnames is generated.

- Input library and path specifications are accepted with or without the `-l` and `-L` prefixes used by Unix linkers.

  An entry of the form `-La:\foo` specifies the `a:\foo` directory to look for the libraries that follow.

  An entry of the form `-lfoo` specifies the library `foo`, which may be spelled differently depending on what kind of compiler you are using. If you are using GCC, it gets translated to `libfoo.a`, but for other win32 compilers, it becomes `foo.lib`. If no files are found by those translated names, one more attempt is made to find them using either `foo.a` or `libfoo.lib`, depending on whether GCC or some other win32 compiler is being used, respectively.

  If neither the `-L` or `-l` prefix is present in an entry, the entry is considered a directory to search if it is in fact a directory, and a library to search for otherwise. The `$Config{lib_ext}` suffix will be appended to any entries that are not directories and don't already have the suffix.

  Note that the `-L` and `-l` prefixes are **not required**, but authors who wish their extensions to be portable to Unix or OS/2 should use the prefixes, since the Unix-OS/2 version of ext() requires them.

- Entries cannot be plain object files, as many Win32 compilers will not handle object files in the place of libraries.

- Entries in `$potential_libs` beginning with a colon and followed by alphanumeric characters are treated as flags. Unknown flags will be ignored.

  An entry that matches `/:nodefault/i` disables the appending of default libraries found in `$Config{perllibs}` (this should be only needed very rarely).

An entry that matches `/:nosearch/i` disables all searching for the libraries specified after it. Translation of `-Lfoo` and `-lfoo` still happens as appropriate (depending on compiler being used, as reflected by `$Config{cc}`), but the entries are not verified to be valid files or directories.

An entry that matches `/:search/i` reenables searching for the libraries specified after it. You can put it at the end to enable searching for default libraries specified by `$Config{perllibs}`.

- The libraries specified may be a mixture of static libraries and import libraries (to link with DLLs). Since both kinds are used pretty transparently on the Win32 platform, we do not attempt to distinguish between them.

- LDLOADLIBS and EXTRALIBS are always identical under Win32, and BSLOADLIBS and LD_RUN_PATH are always empty (this may change in future).

- You must make sure that any paths and path components are properly surrounded with double-quotes if they contain spaces. For example, `$potential_libs` could be (literally):

  ```
  "-Lc:\Program Files\vc\lib" msvcrt.lib "la test\foo bar.lib"
  ```

  Note how the first and last entries are protected by quotes in order to protect the spaces.

- Since this module is most often used only indirectly from extension `Makefile.PL` files, here is an example `Makefile.PL` entry to add a library to the build process for an extension:

  ```
  LIBS => ['-lgl']
  ```

  When using GCC, that entry specifies that MakeMaker should first look for `libgl.a` (followed by `gl.a`) in all the locations specified by `$Config{libpth}`.

  When using a compiler other than GCC, the above entry will search for `gl.lib` (followed by `libgl.lib`).

  If the library happens to be in a location not in `$Config{libpth}`, you need:

  ```
  LIBS => ['-Lc:\gllibs -lgl']
  ```

  Here is a less often used example:

  ```
  LIBS => ['-lgl', ':nosearch -Ld:\mesalibs -lmesa -luser32']
  ```

  This specifies a search for library `gl` as before. If that search fails to find the library, it looks at the next item in the list. The `:nosearch` flag will prevent searching for the libraries that follow, so it simply returns the value as `-Ld:\mesalibs -lmesa -luser32`, since GCC can use that value as is with its linker.

  When using the Visual C compiler, the second item is returned as `-libpath:d:\mesalibs mesa.lib user32.lib`.

  When using the Borland compiler, the second item is returned as `-Ld:\mesalibs mesa.lib user32.lib`, and MakeMaker takes care of moving the `-Ld:\mesalibs` to the correct place in the linker command line.

SEE ALSO

ExtUtils::MakeMaker