NAME

DynaLoader::Functions – deconstructed dynamic C library loading

SYNOPSIS

```
use DynaLoader::Functions qw(
    loadable_for_module
    linkable_for_loadable linkable_for_module);

$loadable = loadable_for_module("Acme::Widget");
@linkable = linkable_for_loadable($loadable);
@linkable = linkable_for_module("Acme::Widget");

use DynaLoader::Functions qw(dyna_load dyna_resolve dyna_unload);

$libh = dyna_load($loadable, {
            require_symbols => ["boot_Acme__Widget"],
        });
my $bootfunc = dyna_resolve($libh, "boot_Acme__Widget");
dyna_unload($libh);
```

DESCRIPTION

This module provides a function-based interface to dynamic loading as used by Perl. Some details of dynamic loading are very platform-dependent, so correct use of these functions requires the programmer to be mindful of the space of platform variations.

FUNCTIONS

File finding

loadable_for_module(MODULE_NAME)

MODULE_NAME must be the name of a Perl module, in bareword syntax with `::` separators. The named module is presumed to be an XS extension following standard conventions, and its runtime-loadable C library file is searched for. If found, the name of the library file is returned. If it cannot be found, the function `dies` with an informative error message.

If the named module is actually not an XS extension, or is not installed, or stores its C library in a non-standard place, there is a non-trivial danger that this function will find some other library file and believe it to be the right one. This function should therefore only be used when there is an expectation that the module is installed and would in normal operation load its corresponding C library.

linkable_for_loadable(LOADABLE_FILENAME)

If symbols in one runtime-loadable C library are to be made available to another runtime-loadable C library, depending on the platform it may be necessary to refer to the exporting library when linking the importing library. Generally this is not required on Unix, but it is required on Windows. Where it is required to refer to the exporting library at link time, the file used may be the loadable library file itself, or may be a separate file used only for this purpose. Given the loadable form of an exporting library, this function determines what is required at link time for an importing library.

LOADABLE_FILENAME must be the name of a runtime-loadable C library file. The function checks what is required to link a library that will at runtime import symbols from this library. It returns a list (which will be empty on many platforms) of names of files that must be used as additional objects when linking the importing library.

linkable_for_module(MODULE_NAME)

Performs the job of "linkable_for_loadable" (which see for explanation), but based on a module name instead of a loadable library filename.

MODULE_NAME must be the name of a Perl module, in bareword syntax with `::` separators. The function checks what is required to link a library that will at runtime import symbols from the loadable C library associated with the module. It returns a list (which will be empty on many platforms) of names of files that must be used as additional objects when linking the importing library.

Low-level dynamic loading

dyna_load(LOADABLE_FILENAME[, OPTIONS])

Dynamically load the runtime-loadable C library in the file named *LOADABLE_FILENAME*. The process is influenced by optional information supplied in the hash referenced by *OPTIONS*. On the platforms that make dynamic loading easiest it is not necessary to supply any options (in which case the parameter may be omitted), but if wide portability is required then some options are required. The permitted keys in the *OPTIONS* hash are:

resolve_using

Reference to an array, default empty, of names of additional library files required to supply symbols used by the library being loaded. On most platforms this is not used. On those platforms where it is required, the need for this will be known by whatever generated the library to be loaded, and it will normally be set by a bootstrap file (see **use_bootstrap_options** below).

require_symbols

Reference to an array, default empty, of names of symbols expected to be found in the library being loaded. On most platforms this is not used, but on some a library cannot be loaded without naming at least one symbol for which a need can be satisfied by the library.

use_bootstrap_options

Truth value, default false, controlling whether a ‘‘bootstrap’’ file will be consulted as an additional source of options to control loading. The ‘‘bootstrap’’ file, if it exists, is located in the same directory as the loadable library file, and has a similar name differing only in its `.bs` ending.

symbols_global

Truth value, default false, indicating whether symbols found in the library being loaded must be made available to subsequently-loaded libraries. Depending on platform, symbols may be so available even if it is not requested. Some platforms, on the other hand, can’t provide this facility.

On platforms incapable of making loaded symbols globally available, currently loading is liable to claim success while leaving the symbols de facto unavailable. It is intended that in the future such platforms will instead generate an exception when this facility is requested.

unresolved_action

String keyword indicating what should be done if unresolved symbols are detected while loading the library. It may be "**ERROR**‘‘ (default) to treat it as an error, ’’**WARN**‘‘ to emit a warning, or ’’**IGNORE**" to ignore the situation. Some platforms can’t detect this problem, so passing this check doesn’t guarantee that there won’t be any runtime problems due to unresolved symbols.

On success, returns a handle that can be used to refer to the loaded library for subsequent calls to ‘‘dyna_resolve’’ and ‘‘dyna_unload’’. On failure, `dies`.

dyna_resolve(LIBRARY_HANDLE, SYMBOL_NAME[, OPTIONS])

Resolve the symbol *SYMBOL* in the previously-loaded library identified by the *LIBRARY_HANDLE*. The process is influenced by optional information supplied in the hash referenced by *OPTIONS*. The permitted keys in the *OPTIONS* hash are:

unresolved_action

String keyword indicating what should be done if the symbol cannot be resolved. It may be "**ERROR**‘‘ (default) to treat it as an error, ’’**WARN**" to emit a warning and return `undef`, or "**IGNORE**" to return `undef` without a warning.

On success, returns the value of the specified symbol, in a platform-dependent format. Returns `undef` if the symbol could not be resolved and this is not being treated as an error.

dyna_unload(LIBRARY_HANDLE[, OPTIONS])

Unload the previously-loaded library identified by the *LIBRARY_HANDLE*. The process is influenced by optional information supplied in the hash referenced by *OPTIONS*. The permitted keys in the *OPTIONS* hash are:

fail_action

String keyword indicating what should be done if unloading detectably fails. It may be "**ERROR**“ (default) to treat it as an error, ”**WARN**“ to emit a warning, or ”**IGNORE**" to ignore the situation.

On some platforms unloading is not possible. On any platform, unloading can be expected to cause mayhem if any code from the library is currently executing, if there are any live references to data in the library, or if any symbols provided by the library are referenced by any subsequently-loaded library.

SEE ALSO

DynaLoader, ExtUtils::CBuilder, XSLoader

AUTHOR

Andrew Main (Zefram) <zefram@fysh.org>

COPYRIGHT

LICENSE