Dynamic Linker

`Libdl.dlopen` — Function

```
dlopen(libfile::AbstractString [, flags::Integer]; throw_error:Bool = true)
```

Load a shared library, returning an opaque handle.

The extension given by the constant `dlext` (`.so`, `.dll`, or `.dylib`) can be omitted from the `libfile` string, as it is automatically appended if needed. If `libfile` is not an absolute path name, then the paths in the array `DL_LOAD_PATH` are searched for `libfile`, followed by the system load path.

The optional flags argument is a bitwise-or of zero or more of `RTLD_LOCAL`, `RTLD_GLOBAL`, `RTLD_LAZY`, `RTLD_NOW`, `RTLD_NODELETE`, `RTLD_NOLOAD`, `RTLD_DEEPBIND`, and `RTLD_FIRST`. These are converted to the corresponding flags of the POSIX (and/or GNU libc and/or MacOS) dlopen command, if possible, or are ignored if the specified functionality is not available on the current platform. The default flags are platform specific. On MacOS the default `dlopen` flags are `RTLD_LAZY|RTLD_DEEPBIND|RTLD_GLOBAL` while on other platforms the defaults are `RTLD_LAZY|RTLD_DEEPBIND|RTLD_LOCAL`. An important usage of these flags is to specify non default behavior for when the dynamic library loader binds library references to exported symbols and if the bound references are put into process local or global scope. For instance `RTLD_LAZY|RTLD_DEEPBIND|RTLD_GLOBAL` allows the library's symbols to be available for usage in other shared libraries, addressing situations where there are dependencies between shared libraries.

If the library cannot be found, this method throws an error, unless the keyword argument `throw_error` is set to `false`, in which case this method returns `nothing`.

`Libdl.dlopen_e` — Function

```
dlopen_e(libfile::AbstractString [, flags::Integer])
```

Similar to `dlopen`, except returns `C_NULL` instead of raising errors. This method is now deprecated in favor of `dlopen(libfile::AbstractString [, flags::Integer]; throw_error=false)`.

`Libdl.RTLD_NOW` — Constant

```
RTLD_DEEPBIND
RTLD_FIRST
RTLD_GLOBAL
RTLD_LAZY
RTLD_LOCAL
RTLD_NODELETE
RTLD_NOLOAD
RTLD_NOW
```

Enum constant for `dlopen`. See your platform man page for details, if applicable.

`Libdl.dlsym` — Function

```
dlsym(handle, sym)
```

Look up a symbol from a shared library handle, return callable function pointer on success.

`Libdl.dlsym_e` — Function

```
dlsym_e(handle, sym)
```

Look up a symbol from a shared library handle, silently return `C_NULL` on lookup failure. This method is now deprecated in favor of `dlsym(handle, sym; throw_error=false)`.

`Libdl.dlclose` — Function

```
dlclose(handle)
```

Close shared library referenced by handle.

```
dlclose(::Nothing)
```

For the very common pattern usage pattern of

```
try
    hdl = dlopen(library_name)
    ... do something
finally
    dlclose(hdl)
end
```

We define a `dlclose()` method that accepts a parameter of type `Nothing`, so that user code does not have to change its behavior for the case that `library_name` was not found.

`Libdl.dlext` — Constant

```
dlext
```

File extension for dynamic libraries (e.g. dll, dylib, so) on the current platform.

`Libdl.dllist` — Function

```
dllist()
```

Return the paths of dynamic libraries currently loaded in a `Vector{String}`.

`Libdl.dlpath` — Function

```
dlpath(handle::Ptr{Cvoid})
```

Given a library `handle` from `dlopen`, return the full path.

```
dlpath(libname::Union{AbstractString, Symbol})
```

Get the full path of the library `libname`.

Example

```
julia> dlpath("libjulia")
```

`Libdl.find_library` — Function

```
find_library(names, locations)
```

Searches for the first library in `names` in the paths in the `locations` list, `DL_LOAD_PATH`, or system library paths (in that order) which can successfully be dlopen'd. On success, the return value will be one of the names (potentially prefixed by one of the paths in locations). This string can be assigned to a `global const` and used as the library name in future `ccall`'s. On failure, it returns the empty string.

`Base.DL_LOAD_PATH` — Constant

```
DL_LOAD_PATH
```

When calling `dlopen`, the paths in this list will be searched first, in order, before searching the system locations for a valid library handle.

« LibGit2 Linear Algebra »