# Dynamic Library Identification

This thread has been locked by a moderator.

210

I often find myself helping folks with dynamic library problems. These problems manifest in many different ways, but they all have one common factor: The dynamic library has an atypical install name. Sometimes this was inherited from macOS's deep past, but most often it's because folks aren't aware of the two well-trodden paths through this particular minefield. This post is my attempt to rectify that.

If you have questions or comments about this, start a new thread here on DevForums. Tag it with *Linker* so that I see it.

Share and Enjoy
—
Quinn "The Eskimo!" @ Developer Technical Support @ Apple
`let myEmail = "eskimo" + "1" + "@" + "apple.com"`

## Dynamic Library Identification

Apple's dynamic linker has a lot of flexibility. Much of this flexibility exists for historical reasons, and it's better for modern programs to follow one of two well-trodden paths:

- Most dynamic libraries should use an rpath-relative install name, as explained in Dynamic Library Standard Setup for Apps.
- In some cases it might make sense to use a full path for your install name, per Dynamic Library Full Path Alternative.

To understand these options, you need to know a little bit about how the dynamic linker works, which is the subject of this post.

**Note** This post covers some of the same ground as Embedding nonstandard code structures in a bundle, but in a less official way (-:

## Install Name

Every dynamic library has an **install name**. This name identifies the library to the dynamic linker. When you link to a dynamic library, the static linker records the library's install name in your Mach-O image. When the dynamic linker loads your Mach-O image, it uses those recorded install names to find and load the libraries you depend on.

**Note** There are *many* different aliases for the term *install name*, including *id*, *identification name*, or *install path*.

To see a library's install name, run `otool` and examine the `LC_ID_DYLIB` load command:

```
% otool -l libWaffle.dylib | grep -A 2 LC_ID_DYLIB
          cmd LC_ID_DYLIB
      cmdsize 48
         name @rpath/libWaffle.dylib …
```

**Note** The Mach-O load commands and their associated structures are defined in `<mach-o/loader.h>`. For example `LC_ID_DYLIB` is associated with the `dylib_command` structure.

To see the install names of the libraries imported by a Mach-O image, run `otool` and examine the `LC_LOAD_DYLIB` load commands:

```
% otool -l libVarnish.dylib | grep -A 2 LC_LOAD_DYLIB
          cmd LC_LOAD_DYLIB
      cmdsize 48
         name @rpath/libWaffle.dylib …
--
          cmd LC_LOAD_DYLIB
      cmdsize 56
         name /usr/lib/libSystem.B.dylib …
```

Alternatively, use the `-L` option:

```
% otool -L libVarnish.dylib
…
    @rpath/libVarnish.dylib …
    @rpath/libWaffle.dylib …
    /usr/lib/libSystem.B.dylib …
```

This displays the library's own install name first, followed by the install names of all the libraries it imports.

If you're building a dynamic library with Xcode, use the Dynamic Library Install Name build setting to set the install name. If you're building from the command line, pass the `-install_name` option to `ld`. For more about this, see the `ld` man page.

It's best to set the install name at build time and not change it. However, if you're working with an existing dynamic library that has the wrong install name, you can usually change it using `install_name_tool`. See the `install_name_tool` man page for details.

## Runtime Path List

Way back in the day, a dynamic library's install name was the full path of the installed library. That's *why* it's called the install name. However, full paths don't work in some situations. For example, if you have a library embedded within an app, you can't use an full path because the app might not be installed in the Applications folder.

To address this problem Apple updated the dynamic linker to support a number of special install name prefixes. At runtime it replaces the prefix with an appropriate path. For example, `@executable_path` is replaced with the path to the directory containing the processes main executable.

For a full list of these prefixes see the `dyld` man page. However, if you're creating a dynamic library you'll want to focus on the runtime path, `@rpath`, or just **rpath** for short. The exact details of how this works are complex, see the man page for the full story, but the basic gist is that:

- The dynamic linker maintains a list of rpath directories.
- When it loads a Mach-O image, the dynamic linker looks for `LC_RPATH` load commands in the image. For each one it finds, it adds a new directory to the rpath list.
- When a Mach-O image imports a library with an rpath-relative install name, the dynamic linker searches for that library in each directory in the list.

This may sound a bit abstract, so you might want to hop on over to Dynamic Library Standard Setup for Apps for some examples of how this works in practice.

If you're building a Mach-O image with Xcode, use the Runpath Search Paths build setting to add rpath directories. If you're building from the command line, pass the `-rpath` option to `ld`. For more about this, see the `ld` man page.

It's best to configure your rpath directories at build time. However, if you're working with an existing Mach-O that has the wrong rpath directories, you can add, delete, and change them using `install_name_tool`. See the `install_name_tool` man page for details.

Linker

Reply

Posted 1 month ago by eskimo

Add a Comment

This site contains user submitted content, comments and opinions and is for informational purposes only. Apple disclaims any and all liability for the acts, omissions and conduct of any third parties in connection with or related to your use of the site. All postings and use of the content on this site are subject to the Apple Developer Forums Participation Agreement.



Copyright © 2023 Apple Inc. All rights reserved. Terms of Use | Privacy Policy | License Agreements