How do I compile and run a Common Lisp program from the directory of the .asd file?

Asked 3 years, 4 months ago Modified 1 year, 11 months ago Viewed 3k times

9

I have the following directory structure:

```
my-project/
├── my-project.asd
├── package.lisp  # defpackage.
├── utils.lisp    # Functions used by main.lisp.
└── main.lisp     # Main program.
```

`my-project.asd` :

```
(defsystem "my-project"
  :components ((:file "package")
               (:file "utils")
               (:file "main")))
```

`package.lisp` :

```
(defpackage :com.example
  (:use :cl))
```

`utils.lisp` :

```
(in-package :com.example)

(defun double (x)
  (* x 2))
```

`main.lisp` :

```
(in-package :com.example)

(format t "~a" (double 3))
```

The problem is: how do I compile and run `main.lisp` using ASDF?

I was able to compile and run the program by:

```
$ mv my-project ~/common-lisp/.
$ sbcl
* (require :asdf)
* (asdf:load-system :my-project)
```

However, this is incredibly silly. I do not want to move my project into `~/common-lisp/` just to run it. I want to compile and run the program right from the project directory itself. The `my-project/` directory could be anywhere, and I want it to be possible to be placed anywhere. In other words, I would like to load the system from the current directory.

Think of `make`, where I can compile files right from the directory of the `Makefile` itself. How do I similarly compile and run a Common Lisp program from the directory of the `*.asd` file itself?

(I am using SBCL version 1.4.5 and ASDF version 3.3.1)

common-lisp asdf

Share Edit Follow Flag

edited Feb 23, 2021 at 21:16

asked Dec 15, 2020 at 6:31
Flux
10.4k 6 52 101

4 Answers

Sorted by: Highest score (default)

8

I found that it is possible to do the following:

```
$ sbcl
* (require "asdf")
* (asdf:load-asd (merge-pathnames "my-project.asd" (uiop:getcwd)))
* (asdf:load-system :my-project)
```

Note:

- `(require "asdf")` is the recommended way to load ASDF, according to the "Loading ASDF" section of the ASDF manual.

  > NB: all implementations except GNU CLISP also accept `(require "ASDF")`, `(require 'asdf)` and `(require :asdf)`. For portability's sake, you should use `(require "asdf")`.

- `asdf:load-asd` must be an absolute path and might not fail with any error when the path given to it is incorrect (!), so make sure that the given absolute path is correct.
- Using `cl:load` instead of `asdf:load-asd` might also appear to work, but the ASDF manual explicitly warns against this practice:

  > Indeed, **ASDF does not load `.asd` files simply with `cl:load`, and neither should you**. You should let ASDF find and load them when you operate on systems. If you somehow must load a `.asd` file, use the same function `asdf:load-asd` that ASDF uses. Among other things, it already binds the

`*package*` to `asdf-user`. Recent versions of SLIME (2013-02 and later) know to do that when you `C-c C-k` when you use the `slime-asdf` contrib.

Share Edit Follow Flag edited May 21, 2022 at 9:35 answered Dec 15, 2020 at 10:50

Flux
10.4k 6 52 101

1 +1. So on one command it gives: `sbcl --load my-project.asd --eval '(asdf:load-system :my-project)'`. This gives you a prompt. You could add a call to `uiop:quit`, or `asdf:make`, etc. – Ehvince Dec 15, 2020 at 11:10

Thank you! This was a very helpful input from you! – João Esperancinha Feb 28 at 9:54

4

This is what I do to solve this problem. It may not be exactly what you need, but there may be ideas here that can help.

For development

Make sure there is only one copy of the system you're trying to build. So, in particular make sure there's no installed copy which ASDF might find instead: see below.

Then what will work is this. Firstly make sure your ASDF system definition is cold-loadable, so in particular make sure it has the right `(in-package :asdf-user)` at the top.

Then what will work to build your system is:

```
$ cd .../my-project
$ sbcl
[...]
* (require :asdf) ;already loaded in my case by init files
nil
* (load "my-project.asd")
t
* (asdf:load-system "my-project")
; compiling file "/media/psf/share/tmp/my-project/package.lisp" (written 15 DEC
2020 09:06:54 AM):
; processing (defpackage :com.
[...]
*
```


And now you're good. So the three tricks I do are:

- avoid thinking about the whole source registry hair at all, because if you think about it too much something with tentacles will tear your face off (I know this, it happened to me, I now have no face);
- make sure there is only one copy of the system so ASDF can't use the source registry hair I have avoided thinking about to find the wrong one;

- explicitly load the system definition file – wherever else ASDF is looking it will at least look in the same directory as that.

For production

The answer to this is Quicklisp. Do whatever is needed to make Quicklisp be installed in your Lisp. Then you need to know where its install directory is: there is some default but I never use that since I have my own notions of what the filesystem should look like.

Then the trick is that *Quicklisp* will find, build and load systems under its `local-projects` directory (Quicklisp is made entirely of competence and magic as far as I can tell). So if you put a system there, then Quicklisp will simply and elegantly deal with getting it into the running image.

To do this installation ... I have makefiles. I know, I should use Lisp tools, but I live on *nix platforms and `make` and `install` are good at the whole copying-files bit.

A relevant chunk of `Makefile` (actually this is really all of it) is:

```
# Where Quicklisp lives, and the name of this project using slashes
QUICKLISP = /local/tfb/packages/quicklisp
THIS      = org/tfeb/sample
FASLS     = *fasl *fsl

.PHONY: install uninstall clean

# To install the project make its directory, copy all the sources and
# the sysdcl into it, and then nuke Quicklisp's cache file so it searches
# next time
#
install:
    @mkdir -p "$(QUICKLISP)/local-projects/$(THIS)"
    @cd "$(QUICKLISP)/local-projects/$(THIS)" && rm -f $(FASLS)
    @install -C -v -m 444 *.lisp *.asd "$(QUICKLISP)/local-projects/$(THIS)"
    @rm -f "$(QUICKLISP)/local-projects/system-index.txt"

# To uninstall the project just nuke the directory and the cache
#
uninstall:
    @rm -rf "$(QUICKLISP)/local-projects/$(THIS)"
    @rm -f "$(QUICKLISP)/local-projects/system-index.txt"

clean:
    @rm -f $(FASLS) *~
```

There are four interesting things here:

- I'm just using `make` as a file-copying machine – it's not compiling anything or anything like that, and it would be perfectly possible to use a script;
- you need to blow away Quicklisp's cache file so it searches again on start;
- I disable ASDF's output translations which is why I spend time blowing away the compiled files – after an install the project always should be rebuilt from cold;

- the `uninstall` target is what you need to run before development – it will nuke the installed version so ASDF doesn't find it.

Once you've run a suitable `make install` in your project's directory, then `(ql:quickload :org.tfeb.sample)` will just compile and load it for you.

Note that an alternative approach (suggested by Ehvince in a comment) would be to leave a symbolic link under Quicklisp's `local-projects` directory back to the canonical version of the code. I don't do that but it would work fine, and might be better in some ways.

Share Edit Follow Flag edited Dec 16, 2020 at 10:19 answered Dec 15, 2020 at 9:44
user5920214

1 A symlink in Quicklisp's local-projects also works. – Ehvince Dec 15, 2020 at 17:45

@Ehvince: yes, that's a good point – I've added a note to the answer in case the comment goes away – user5920214 Dec 16, 2020 at 10:17

1

You need to tell asdf where to find your project.

Here's the relevant reference:

https://common-lisp.net/project/asdf/asdf/Configuring-ASDF-to-find-your-systems.html

Quoting from the above resource:

> First create the directory ~/.config/common-lisp/source-registry.conf.d/; there create a file with any name of your choice but with the type conf, for instance 50-luser-lisp.conf; in this file, add the following line to tell ASDF to recursively scan all the subdirectories under /home/luser/lisp/ for .asd files: (:tree "/home/luser/lisp/")
>
> That's enough. You may replace /home/luser/lisp/ by wherever you want to install your source code. You don't actually need to specify anything if you use the default ~/common-lisp/ as above and your implementation provides ASDF 3.1.2 or later. If your implementation provides an earlier variant of ASDF 3, you might want to specify (:tree (:home "common-lisp/")) for bootstrap purposes, then install a recent source tree of ASDF under ~/common-lisp/asdf/.

There's much more at the link.

Share Edit Follow Flag answered Dec 15, 2020 at 7:26
Capstone
2,284 2 20 39

Should I tell ASDF to recursively scan from the root of the my home directory (~) if `my-project/` could be anywhere? – Flux Dec 15, 2020 at 10:59

@Flux No, don't do that! – Ehvince Dec 15, 2020 at 11:32

I don't think messing with the source registry is necessary, especially when we are in the same directory of the .asd (since we can `--load` it). – Ehvince Dec 15, 2020 at 11:32

0

I do something similar as you found out. The gist of it is to `--load` the .asd.

Prerequisites

`my-project.asd` starts with `(require "asdf")`.

```
(require "asdf")
(asdf:defsystem "my-project"
  :version "0.0"
  …)
```

When I am on Slime, I can `C-c C-k` (compile and load) this file.

I am not so sure if/why it is required for "--load".

One-liner

I can build the project with one invocation of the Lisp compiler:

```
sbcl --load my-project.asd --eval '(asdf:load-system "my-project")'
```

This loads it and gives me a REPL. I could add `--eval (uiop:quit)` to quit.

Note: I heard people saying it's best to use `asdf:load-asd`.

With Quicklisp - necessary when you have dependencies

Instead of `asdf:load-system`, I actually use Quicklisp, because it will load my project's dependencies.

```
sbcl --load my-project.asd --eval '(ql:quickload "my-project")'
```

(Note that I didn't copy my project to Quicklisp's local-projects. If I did, I wouldn't need to load the .asd here)

With a Makefile

This one-liner can be turned into a simple Makefile.

```
LISP ?= sbcl
build:
    $(LISP) --load my-project.asd \
         --eval '(ql:quickload :my-project)' \
         # more rules here
         --eval '(quit)'
```

Simplifying with a lisp file to run them all

We need to:

1- load the .asd

2- quickload the dependencies

3- run our script

We can also do this from a lisp file.

`run.lisp` :

```
(load "my-project.asd")  ;; relative path: we must be in the same directory
(ql:quickload "my-project")  ;; I installed Quicklisp and my ~/.sbclrc has the
Quicklisp-initialization snippet

(my-project::main)  ;; whatever function acts as the entrypoint
(uiop:quit)         ;; portable way to quit.
```

Building a binary

I use `asdf:make` as explained here: https://lispcookbook.github.io/cl-cookbook/scripting.html#with-asdf

Share Edit Follow Flag

answered Dec 15, 2020 at 11:27

Ehvince

17.8k 7 60 81