

---

## lein-cljsbuild

**Latest version:**

```
[lein-cljsbuild "1.1.8"]
```

@clojars.org

This is a Leiningen plugin that makes it quick and easy to automatically compile your ClojureScript code into Javascript whenever you modify it. It's simple to install and allows you to configure the ClojureScript compiler from within your `project.clj` file.

Beyond basic compiler support, lein-cljsbuild can optionally help with a few other things:

- Launching REPLs for interactive development
- Launching ClojureScript tests

The latest version of lein-cljsbuild is 1.1.8. See the release notes here.

*Note that cljsbuild crossovers are deprecated, and will be removed eventually. You should never use them. Please use either reader conditionals (available in Clojure >= 1.7.0-beta2 and ClojureScript >= 0.0-3255), or cljx to target both Clojure and ClojureScript from the same codebase.*

**If you are using ClojureScript >= 1.7.170 you need to use a lein-cljsbuild version >= 1.1.1.**

### Requirements

The lein-cljsbuild plugin works with Leiningen version 2.1.2 or higher.

### Installation

You can install the plugin by adding lein-cljsbuild to your `project.clj` file in the `:plugins` section:

```
1 (defproject lein-cljsbuild-example "1.2.3"
2   :plugins [[lein-cljsbuild "1.1.8"]])
```

In addition, you should add an explicit ClojureScript dependency to your project using the ClojureScript version you want to use:

```
1 :dependencies [[org.clojure/clojurescript "1.9.521"]]
```

lein-cljsbuild will add a dependency to your project if it doesn't already contain one, but that functionality will not remain for long. The latest version of lein-cljsbuild currently requires a minimum of ClojureScript 0.0-3211.

---

## Just Give Me a Damned Example Already!

See the `example-projects` directory for a couple of simple examples of how to use `lein-cljsbuild`. The simple project shows a dead-simple “compile only” configuration, which is a good place to start. The advanced project contains examples of how to use the extended features of the plugin.

For an exhaustive list of all options supported by `lein-cljsbuild`, see the `sample.project.clj` file. For a list of options that can be passed to the ClojureScript compiler have a look at the ClojureScript Compiler Options site.

## Basic Configuration

The `lein-cljsbuild` configuration is specified under the `:cljsbuild` section of your `project.clj` file. A simple project might look like this:

```
1 (defproject lein-cljsbuild-example "1.2.3"
2   :plugins [[lein-cljsbuild "1.1.8"]]
3   :cljsbuild {
4     :builds [{
5       ; The path to the top-level ClojureScript source directory:
6       :source-paths ["src-cljs"]
7       ; The standard ClojureScript compiler options:
8       ; (See the ClojureScript compiler documentation for details.)
9       :compiler {
10        :output-to "war/javascripts/main.js" ; default: target/
11        :optimizations :whitespace
12        :pretty-print true}}]})
```

## Basic Usage

Once the plugin is installed, you can build the ClojureScript once:

```
1 $ lein cljsbuild once
```

Or you can have `lein-cljsbuild` watch your source files for changes and automatically rebuild them. This is recommended for development, as it avoids the time-consuming JVM startup for each build:

```
1 $ lein cljsbuild auto
```

Assuming you have configured `cljsbuild` to emit compiler output to one of Leiningen’s `:clean-targets` (which includes `./target` by default), running `lein clean` will delete all of the JavaScript and ClojureScript files that `lein-cljsbuild` generates during compilation.

---

## Color-coded output on Windows

Colors are a big deal when reading ClojureScript compiler output, but Windows consoles don't support ANSI color codes. This limitation is commonly corrected by installing ANSICON:

1. Download and unzip ANSICON anywhere.
2. Open a command prompt (Run as administrator).
3. Navigate to the unzipped folder.
4. `cd x86` or `x64` (depending on whether you have 32-bit or 64-bit machine, respectively)
5. Run `ansicon -i` to install.

Afterwards, you should get colored output from all future console sessions that use ANSI color codes.

## Hooks

Some common `lein-cljsbuild` tasks can hook into the main Leiningen tasks to enable ClojureScript support in each of them. The following tasks are supported:

```
1 $ lein compile
2 $ lein test
3 $ lein jar
```

To enable ClojureScript support for these tasks, add the following entry to your project configuration:

```
1 :hooks [leiningen.cljsbuild]
```

Note that by default the `lein jar` task does *not* package your ClojureScript code in the JAR file. This feature needs to be explicitly enabled by adding the following entry to each of the `:builds` that you want included in the JAR file. `lein uberjar` derives its behavior from `lein jar` and will include the ClojureScript as well if enabled.

```
1 :jar true
```

**Debug Note:** There is a known issue (#366) where the `lein uberjar` task fails to build when using hooks and a `cljsbuild` configuration within an `:uberjar` profile. Instead of hooks, you can use `:prep-tasks` as an alternative:

```
1 :prep-tasks ["compile" ["cljsbuild" "once"]]
```

---

## Multiple Build Configurations

If the `:builds` sequence contains more than one map `lein-cljsbuild` will treat each map as a separate ClojureScript compiler configuration, and will build all of them in parallel:

```
1 (defproject lein-cljsbuild-example "1.2.3"
2   :plugins [[lein-cljsbuild "1.1.8"]]
3   :cljsbuild {
4     :builds [
5       {:source-paths ["src-cljs-main"]
6        :compiler {:output-to "main.js"}}
7       {:source-paths ["src-cljs-other"]
8        :compiler {:output-to "other.js"}}])})
```

This is extremely convenient for doing library development in ClojureScript. This allows `cljsbuild` to compile in all four optimization levels at once, for easier testing, or to compile a test suite alongside the library code.

You can optionally assign an ID to a build configuration and build only that one:

```
1 (defproject lein-cljsbuild-example "1.2.3"
2   :plugins [[lein-cljsbuild "1.1.8"]]
3   :cljsbuild {
4     :builds [
5       {:source-paths ["src-cljs-main"]
6        :compiler {:output-to "main.js"}}
7       {:id "other"
8        :source-paths ["src-cljs-other"]
9        :compiler {:output-to "other.js"}}])})
```

```
1 $ lein cljsbuild auto other
```

If you want IDs for all of your build configurations, you can specify them as a map instead of a vector:

```
1 (defproject lein-cljsbuild-example "1.2.3"
2   :plugins [[lein-cljsbuild "1.1.8"]]
3   :cljsbuild {
4     :builds {
5       :main
6       {:source-paths ["src-cljs-main"]
7        :compiler {:output-to "main.js"}}
8       :other
9       {:source-paths ["src-cljs-other"]
10        :compiler {:output-to "other.js"}}})})
```

You can also build multiple configurations at once:

```
1 $ lein cljsbuild auto main other
```

---

See the `example-projects/advanced` directory for a working example of a project that uses this feature.

## REPL Support

Lein-cljsbuild has built-in support for launching ClojureScript REPLs in a variety of ways. See the REPL documentation for more details.

## Testing Support

Lein-cljsbuild has built-in support for running external ClojureScript test processes. See the testing documentation for more details.

## Extended Configuration

### Custom warning handlers

You can place custom warning handlers for the ClojureScript compiler under the `:warning-handlers` key. The value should be a vector of either 1.) fully-qualified symbols that resolve to your custom handler, or 2.) anonymous functions that will get eval'd at project build-time.

```
1 (defproject lein-cljsbuild-example "1.2.3"
2   :plugins [[lein-cljsbuild "1.0.4"]]
3   :cljsbuild {
4     :builds {:id "example"
5               :compiler {}
6               :warning-handlers [my.ns/custom-warning-handler ;; Fully-
                               qualified symbol
7
                               ;; Custom function (to be evaluated at
                               project build-time)
8               (fn [warning-type env extra]
9                 (when-let [s (cljs.analyzer/error-
10                             message warning-type extra)]
11                   (binding [*out* *err*]
                     (cljs.analyzer/message env s))))
12               ]})
```

## ClojureScript Version

After configuring lein-cljsbuild, `lein deps` will fetch a known-good version of the ClojureScript compiler. You can use a different version of the compiler via a local clone of the ClojureScript git repository.

---

See the wiki for details.

## **License**

Source Copyright © Evan Mezeske, 2011-2013. Released under the Eclipse Public License - v 1.0. See the file COPYING.

## **Contributors**

A big thank you to all contributors who help to make this project better.