# SML/NJ CM Usage

Parallel and Sequential Data Structures and Algorithms, 15-210 (Spring 2014)

*January 13th, 2014*

## 1 The `make` function

Managing code that lives in several SML files–loading the files in the correct order and avoiding shadowing issues as the program grows–becomes irritating quickly. In order to avoid tedious and error-prone sequences of `use` commands, the authors of SML/NJ included a program that will load and compile the source code for programs from a text file that lists the files which contain it.

The structure `CM` has a function:

```
val make : string -> unit
```

This function takes the name of the listing text file, a file usually named `sources.cm`.

## 2 Sources

The file `sources.cm` usually has the following form:

```
Group is
  $/basis.sml
  file1.sml
  file2.sml
  file3.sml
```

Note that `CM` will not run if any of the files specified are empty, or contain no module definitions. So you may occasionally need to add files to `sources.cm` yourself, or uncomment lines that we include.

Also note that while `CM` does compile all the code found in the files specified, it only introduces bindings into the *top-level* environment for values, types, and exceptions defined *within* modules. All other bindings are discarded.

## 3 Using `CM` in the REPL

Once you have `sources.cm` set up, loading your code using the *read-eval-print-loop* (REPL) is simple. Launch SML in the directory containing your code, and call `CM.make "sources.cm"`:

```
$ smlnj
Standard ML of New Jersey v110.xx
```

```
- CM.make "sources.cm";
 [autoloading]
 [library $smlnj/cm/cm.cm is stable]
 [library $smlnj/internal/cm-sig-lib.cm is stable]
 ...
 ...
 [New bindings added.]
 val it = true : bool
-
```

That last "-" prompt, is an indication that the bindings are ready. You may now test your code using the `Tester` structure we've provided to you. Call `CM.make "sources.cm"` again when you've changed something in your code and want to recompile.

## 4 Why use CM?

The compilation manager offers a better interface to the command line. There is less typing and less of an issue with name-shadowing between iterations of your code. In short, when using `CM`, your development cycle will look something like:

1. Edit your sources files.
2. At the REPL, type

   ```
   CM.make "sources.cm";
   ```

   or if you're in `smlnj` (which is simply a wrapper on top of `sml` with `rlwrap`, which enables you to view command history with the arrow keys), just use the UP and DOWN arrows.
3. Fix errors and debug.
4. If done, celebrate! Otherwise, go back to step 1.

Calling `CM.make` will make a hidden subdirectory in the current working directory called `.cm`. This is populated with metadata needed to work out compilation dependencies, but can become quite large. Since it's entirely generated, the `.cm` directory can be safely deleted between work sessions.

## 5 Further Help

An extremely detailed discussion of `CM` and its implementation in SML/NJ can be found at:

http://www.smlnj.org/doc/CM/new.pdf

As always, don't hesitate to ask a TA for help if you're unclear about anything!