CS 153 **Foundations of Software** **Handout 3**
Fall 2006 **11/25/06**

Loading Source Files in SML

In Homework 7 there were typically two SML source files to be loaded. The file "`wff-mod.sml`" contains the WFF module for the wff datatype, StringSet for string sets (used to model sets of variables), Semant for evaluating the truth of wffs relative to an interpretation, and Tautology for testing validity and satisfiability. Another file, which we will assumed is named "`pc.sml`", contained the solution.

How do we load and execute a program like this that consists of more than one source file? The simplest way to load files in the SML/NJ interactive system (at the "-" prompt that you get when you run sml), is to call the `use` function, passing it the file name or path as a string. So to load two files we can call `use` twice, passing the appropriate file names.

```
- use "wff-mod.sml";
- use "pc.sml";
```

The effect of `use` is to process the declarations in the source files as though the contents of the files had been typed into the interactive top-level loop. As each call of `use` is executed, the system will print out the usual top-level response indicating successful compilation and execution or the detection of syntax or type errors.[1] The accidentally extended expression will continue until either a semicolon is encountered or a keyword (e.g. `structure`) not allowed in an expression causes a syntax error.

Instead of manually loading each file by typing these commands in the interactive system, one could place the line

```
use "wff-mod.sml";
```

at the beginning of the `pc.sml` file. Then when `pc.sml` file was loaded it would automatically execute this embedded call of `use` to load `wff-mod.sml`. But this is not the recommended way of linking files together. Instead, we could partially automate the loading of multiple files by creating another SML source file `files.sml` that would specify what files are loaded and in what order. This file would contain just the two lines:

```
use "wff-mod.sml";
use "pc.sml";
```

Then when this file is loaded by the command

```
- use "files.sml";
```

in the interactive loop, the two `use` commands will be executed, loading the two program source

[1] The semicolons terminating these top-level calls are important. At the top-level, the semicolon is a signal that a "compilation unit" or command is complete and ready to be compiled and executed. Without the semicolon after `"wff-mod.sml"`, the following tokens `use` and `"pc.sml"` would be taken as additional arguments for the first `use` call and `use "wff-mod.sml" use "pc.sml"` would be treated as a single curried call with three arguments.

files.

Building a program with the `use` command is reasonable for small programs a few source files, but as the scale of a project increases, it soon makes more sense to use the CM (Compilation Manager) tool that is build into SML/NJ. To use CM, you begin by defining a CM description file specifying a group of source files to be compiled and loaded. In this case we could name the description file `pc.cm` and it could contain the following lines:

```
Group is

wff-mod.sml
pc.sml

$/basis.cm $/smlnj-lib.cm
```

The last two lines are boiler-plate library specifications that make available the standard Basis Library containing basic modules like Int, String, and List, and the SMLNJ Library of common data structures and other utilities, such as sorting routines. The order in which the file names are specified does not matter, since CM analyzes the dependencies between source files and compiles them in the right order.

Assuming the source files `wff-mod.sml` and `pc.sml` and the CM description file `pc.cm` are all found in the current directory, you can run

```
$ sml
- CM.make "pc.cm";
```

and your files will be compiled (if necessary), loaded, and executed. Binary versions of the compiled files will be stored in a directory named `.cm` for future use, and they won't need to be compiled again, unless something changes in their dependency chain. For instance, if you edit `pc.sml` and call `CM.make` again by typing

```
- CM.make "pc.cm";
```

only the changed file `pc.sml` will be recompiled, and not `wff-mod.sml`. This smart recompilation feature of CM saves a great deal of time in large projects with hundreds or thousands of source files.

CM has many additional features, such as invoking other software tools besides the compiler (includiing, for instance, ml-yacc and ml-lex), and the ability to build binary only archives of libraries. See the CM manual at `http://www.smlnj.org/doc/CM/new.pdf` for further details.