Editing and running Standard ML under GNU Emacs

SML-MODE Version 3.2
July 1995

Author: (see Chapter 5 [Credits], page 16)

1 Introduction

SML mode is a major mode for Emacs for editing Standard ML. It has some novel bugs, and some nice features:

- Automatic indentation of sml code—a number of variables to customise the indentation.
- Forms insertion for commonly used structures, like let, local and signature declarations, with minibuffer prompting for types and expressions.
- Magic pipe insertion. `|` automatically determines if it is used in a case or fun construct, and indents the next line as appropriate, inserting `=>` or the name of the function.
- Inferior shell for running ML. There's no need to leave Emacs, just keep on editing while the compiler runs in another window.
- Automatic "use file" in inferior shell—you can send files, buffers, or regions of code to the ML subprocess.
- Parsing errors from the inferior shell, and repositioning the source—like the next-error function used in c-mode.
- Menus, and syntax and keyword highlighting supported for Emacs 19 and derivatives.

1.1 The SML Mode Distribution

The distribution contains several files of Emacs lisp—this is for ease of maintenance, you can concatenate them if you're careful:

'`sml-mode.el`'
: Main file, and should work in any Emacs editor or version post 18.58—it only knows, or thinks it knows, about SML syntax and indentation.

'`sml-proc.el`'
: Process interaction requires the '`comint`' package—this is distributed with Emacs 19 and XEmacs.

'`sml-{hilite,font}.el`'
: Define syntax highlighting functions to display keywords in a bold font, comments in italics, etc.—require the '`hilit19`' package, currently distributed with Emacs 19, or '`font-lock`' (default under XEmacs).

'`sml-menus.el`'
: Menus to access user settable features of the mode, and for those who prefer menus over keys—requires '`auc-menu`' (or '`easymenu`' supplied with GNU Emacs).

‘sml-{smlnj,poly-ml}.el’

Auxiliary support for SML/NJ and Poly/ML compilers, respectively.

as well as this TEXinfo info file:

‘sml-mode.{dvi,info}’

This file—rudimentary SML mode documentation.

1.2 Getting Started

The main part of SML mode is in the file ‘sml-mode.el’. You will have to tell Emacs where to find this, and the other ‘.el’ files, and when to use them. The where is addressed by locating the lisp code on your Emacs lisp path—you may have to create a directory for this, say ‘/usr/me/elisp’, and then insert the following lines in your ‘/usr/me/.emacs’ configuration file:

```
(setq load-path (cons "/usr/me/elisp" load-path))
(autoload 'sml-mode "sml-mode" "Major mode for editing SML." t)
```

The first line adjusts Emacs' internal search path so it can locate the lisp source you have copied to that directory, the second line tells Emacs to load the code automatically when it is needed. You can then switch any Emacs buffer into SML mode by entering the command

```
M-x sml-mode
```

It is usually more convenient to have Emacs automatically place the buffer in SML mode whenever you visit a file containing ML source. The simplest way of achieving this is to put something like

```
(setq auto-mode-alist
      (append '(("\\.sml$" . sml-mode)
                ("\\.ML$"  . sml-mode)) auto-mode-alist))
```

also in your ‘.emacs’ file. Subsequently (after a restart), any files with these extensions will be placed in SML mode buffers when you visit them.

You may want to pre-compile the ‘sml-*.el’ files (M-x byte-compile-file) for greater speed—byte compiled code loads and runs somewhat faster. Note, though, that you only need to set up the file ‘sml-mode.el’ to autoload from your ‘.emacs’: other functions will autoload from that file.

1.3 Help!

You are reading it. Apart from the on-line `info` tree (`C-h i` is the Emacs key to enter the `info` system—you should follow the tutorial if this is unfamiliar), there are further details on specific commands in their documentation strings. Not all SML mode commands are documented in the info tree, only the most useful ones. To find out more, use Emacs' help facilities.

Briefly, to get help on a specific function use `C-h f` and enter the command name. All (almost all, then) SML mode commands begin with '`sml-`', so if you type this and press TAB (for completion) you will get a list of all commands. Another way is to use `C-h a` and enter the string '`sml`'. This is command apropos; it will list all commands with that sub-string in their names, and any key binding they may have. Command apropos gives a one-line synopsis of what each command does.

Note: some commands are also variables—such things are allowed in lisp, if not in ML! See [Command Index], page i, for a list of (info) documented functions, and see [Variable Index], page i, for a list of user settable variables to control the behaviour of SML mode.

Before accessing this information on-line from within Emacs you may have to set the SML mode variable `sml-mode-info`. Something like:

```
(setq sml-mode-info "/usr/me/info/sml-mode")
```

When different from the default (`"sml-mode"`) this should be a string giving the absolute name of the '`.info`' file. Then `C-c C-i` in SML mode (i.e., the `command sml-mode-info`) will bring up the manual. This help is also accessible from the SML mode menu. (Resetting this variable will not be necessary if your site administrator has been kind enough to install SML mode and its attendant documentation in the Emacs hierarchy.)

2 Editing with SML Mode

SML mode only provides a few additional editing commands. Most of the work (see Chapter 5 [Credits], page 16) has gone into implementing the indentation algorithm which, if you think about it, has to be very complicated for a language like ML. See Section 2.4 [SML Mode Defaults], page 6, for details on how to control some of the behaviour of the indentation algorithm.

Principal goodies are the 'electric pipe' feature, and the ability to insert common SML forms—see Section 2.3 [Magic Insertion], page 5, below.

2.1 On Entering SML Mode

`sml-mode`

This command will switch the buffer into SML mode. Since this is a major Emacs editing mode the buffer's major mode must be set to something else to get out of SML mode. See Section 1.2 [Getting Started], page 2, for details on how to set this up automatically on visiting an ML source file. There are hooks:

`sml-mode-hook`

This is run every time a new SML mode buffer is created. This is one place to put your preferred key bindings (see Chapter 4 [Configuration], page 12, for some examples).

`sml-load-hook`

This is another place for keybindings. This hook is only run when SML mode is loaded into Emacs. (see Section 4.3.2 [Hilites], page 15, for sample usage).

`sml-mode-version`

Prints the current version of SML mode in the mini-buffer, in case you need to know.

2.2 Automatic Indentation

ML is a complicated language to parse, let alone compile. The indentation algorithm is a little wooden (for some tastes), so the best advice is not to fight it! There are several variables that can be adjusted to control the indentation algorithm (see Section 2.4 [SML Mode Defaults], page 6, below). Syntax errors may sometimes confuse indentation—a useful feature, perhaps?

`sml-indent-line`

On `TAB`, by default.

This command indents the current line using the indentation algorithm. If you set the indentation of the previous line by hand, `sml-indent-line` will indent relative to your wishes.

`sml-indent-region`

On `C-M-\`, by default.

Indent the current region. Be patient if the region is large (like the whole buffer).

`sml-back-to-outer-indent`

On `M-TAB`, by default.

Unindents the line to the next outer level of indentation.

Further indentation commands that Emacs provides (generically, for all modes) that you may like to recall:

`newline-and-indent`

On `LFD`, by default.

Insert a newline, then indent according to major mode. See section "Indentation for Programs" in The Emacs Editor Manual, for details.

`indent-for-comment`

On `M-;`, by default.

Indent this line's comment to comment column, or insert an empty comment. See section "Manipulating Comments" in The Emacs Editor Manual.

`indent-new-comment-line`

On `M-LFD`, by default.

Break line at point and indent, continuing comment if within one. See section "Manipulating Comments" in The Emacs Editor Manual.

As with other language modes, `M-;` gives you a comment at the end of the current line. The column where the comment starts is determined by the variable `comment-column`—default is 40, but it can be changed with `set-comment-column` (on `C-x ;` by default).

2.3 Electric Features

Electric keys are generally pretty irritating, so those provided by SML mode are fairly muted. The only truly electric key is `;`, and this has to be enabled to take effect.

`sml-electric-pipe`

On `M-|`, by default.

When the insertion point is in a case-statement this will open a new line, indent and insert `| =>`, and leave point just before the double arrow. If the enclosing construct is a function instead, the newline is indented and the function name copied at the appropriate column. Generally, try it whenever a `|` is wanted—you'll like it!

`sml-electric-semi`

On `;`, by default.

This command is governed by the variable `sml-electric-semi-mode`. When true, `;` inserts a semi-colon, a new line and indents; otherwise just a semi-colon (default).

`sml-insert-form`

On `C-c RET`, by default.

Interactive short-cut to insert a common ML form. Recognised forms are `abstraction`, `abstype`, `case`, `datatype`, `functor`, `let`, `local`, `signature`, and `structure`. Except for let and local, these will prompt for appropriate parameters like functor name and signature, etc.. This command prompts in the mini-buffer, with keyword completion.

The behaviour of `;` is controlled by `sml-electric-semi-mode`. The function of the same name can be used to toggle electric semi mode—a prefix argument will always disable the feature.

2.4 Customising Editing Mode

Several variables control indentation (and other features of SML mode). For these user settable variables there is generally a function of the same name that does the job. To control the indentation algorithm:

`sml-indent-level`

Default: '`4`'.

This variable controls the block indentation level. The function prompts for a numeric value unless a numeric prefix is provided instead.

`sml-pipe-indent`

Default: '`-2`'.

This variable adjusts the indentation level for lines that begin with a `|` character. The extra offset is usually negative. The function prompts for a numeric value unless a numeric prefix is provided instead.

`sml-paren-lookback`

Default: '`1000`'.

The number of characters the indentation algorithm searches for an opening parenthesis. 1000 characters is 30-40 lines; larger values mean slower indentation, and nil means do not look back at all.

If the default values are not acceptable, you can set these variables in your '`.emacs`' file. See Chapter 4 [Configuration], page 12, for details and examples.

Three further variables control the behaviour of indentation. They don't have any effect on `sml-insert-form` however:

`sml-case-indent`

Default: '`nil`'.

How to indent case-of expressions:

If t:
```
case expr
  of exp1 => ...
   | exp2 => ...
```

If nil:
```
case expr of
    exp1 => ...
  | exp2 => ...
```

The first seems to be the standard in SML/NJ. The second is the default.

`sml-nested-if-indent`

Default: '`nil`'.

Nested 'if-then-else' expressions will have the following indentation depending on the value.

If t:
```
if exp1 then exp2
else if exp3 then exp4
else if exp5 then exp6
     else exp7
```

If nil:
```
if exp1 then exp2
else if exp3 then exp4
     else if exp5 then exp6
          else exp7
```

`sml-type-of-indent`

Default: '`t`'.

Determines how to indent 'let', 'struct', etc..

If t:
```
fun foo bar = let
                  val p = 4
              in
                  bar + p
              end
```

If nil:
```
fun foo bar = let
    val p = 4
in
    bar + p
end
```

Will not have any effect if the starting keyword is first on the line.

3 Interacting with Standard ML through Emacs

The most useful feature of SML mode is that it provides a convenient interface to the compiler. How serious users of ML put up with a teletype interface to the compiler is beyond me... but perhaps there are other bolt-on interfaces to the current swatch of ML compilers that require one to part with real money. Such remarks can quickly become dated—in this case, let's hope so!

SML mode provides a rudimentary interaction mode where the compiler runs in a separate buffer—the mode is called `inferior-sml-mode`. One can use this buffer just like a terminal, but usually one marks text in the SML mode buffer and has Emacs communicate with the sub-process.

The features discussed below are syntax-independent, so they should work with a wide range of ML-like tools and compilers. See Section 3.4 [Process Defaults], page 10, for some hints.

`inferior-sml-mode` is a specialisation of the '`comint`' package that comes with GNU Emacs and GNU XEmacs.

3.1 Running the Compiler

Start your favourite ML compiler with the command

```
M-x sml
```

This creates a process interaction buffer that inherits key bindings from SML mode and from '`comint`' (see section "Shell Mode" in The Emacs Editor Manual, for details). Starting the ML compiler adds some functions to SML mode buffers so that program text can be communicated between editor and compiler (see Section 3.2 [Sending Text], page 9, below).

The name of the ML compiler is the first thing you should know how to set. The variable `sml-program-name` is a string holding the name of the program as you would type it at the shell. By default it is set to '`sml`', but you can choose a different name by invoking

```
C-u M-x sml
```

With the prefix argument Emacs will prompt for the command name and any command line arguments to pass to the compiler. Thereafter, Emacs will use this new name as the default, but for a permanent change you should set this in your '`.emacs`' with a declaration like

```
(setq sml-program-name "nj-sml")
```

You probably shouldn't set this in a hook because that will interfere if you occasionally run a different compiler (e.g., `poly` or `hol90`).

`sml` Not bound by default.

If an ML process already exists this switches to the process buffer. A prefix argument allows you to edit the command line. The command runs `comint-mode-hook` and `inferior-sml-mode-hook` hooks in that order, but after the compiler is started.

`switch-to-sml`

On `C-c C-s`, by default.

Switch from SML buffer to the interaction buffer. By default the point will be placed at the end of the process buffer, but a prefix argument will leave point wherever it was before. This command will split the screen (Emacs window) so as to display the source and interaction buffers simultaneously.

`sml-cd` Not bound by default.

When started, the ML compiler's default working directory is the current buffer's default directory. This command allows the working directory to be changed, if the compiler can do this. The variable `sml-cd-command` specifies the compiler command to invoke (see Section 3.4 [Process Defaults], page 10, for details).

It's unlikely you will ever need this, but `inferior-sml-mode` is the command that will put the current buffer into ML interaction mode. Note that if you try `C-c C-s` before an ML process has been started, you'll just get an error message to the effect that there's no current process buffer.

3.2 Sending SML definitions to ML

Several commands are defined for sending program text to the running compiler. Each of the following commands takes a prefix argument that will switch the input focus to the process buffer (leaving point at the end of the buffer):

`sml-load-file`

On `C-c C-l`, by default.

Send a 'use file' command to the ML process. The variable `sml-use-command` is used to define the correct template for the command to invoke (see Section 3.4 [Process Defaults], page 10, below). The default file is the file associated with the current buffer, or if point is in the interaction buffer, the last file sent.

`sml-send-region`

On `C-c C-r`, by default.

Send the current region of text in the SML buffer.

`sml-send-function`

Not bound by default.

Send the enclosing 'function' definition. Contrary to the suggestive name, this command does not try to determine the extent of the function definition because that is too difficult with ML. Instead this just sends the enclosing paragraph (delimited by blank lines or form-feed characters).

`sml-send-buffer`

On `C-c C-b`, by default.

Send the contents of the current buffer to ML.

Two further commands are defined for you to bind to keys if you wish: `sml-send-region-and-go` and `sml-send-function-and-go`. Both automatically switch to the interaction buffer.

3.3 Tracking Syntax Errors

SML mode provides one easily customisable function for locating the source position of errors reported by the compiler. It doesn't matter if you type '`use "myfile.ml";`' into the interaction buffer, or use the mechanisms for sending text directly to the compiler that the mode provides—see Section 3.2 [Sending Text], page 9.

`sml-next-error`

On `C-c'`, by default.

Jump to the source location of the next error reported by the compiler.

`sml-skip-errors`

Not bound by default.

Skip past all remaining error messages in the compiler's output.

Note that Emacs will do an implicit `sml-skip-errors` before issuing a 'use file' (or operating on a temporary file), so the errors located will always be relative to the last major piece of text sent.

If you don't use the temporary file mechanism to communicate text to the ML process (see Section 3.4 [Process Defaults], page 10, below), `sml-skip-errors` won't be run, and errors will not be located on '`std_in`'. This may change.

3.4 Customising Process Interaction

By and large, GNU Emacs can nowadays quite happily send large chunks of text to its subprocesses ('`comint`' does input splitting). However, it is still probably safest[1] to send larger pieces

[1] WARNING: XEmacs users note that changing the default value of `sml-temp-threshold` will probably cause v19.11 to hang. A likely XEmacs error, seems fixed in v19.12.

of text, especially if they contain use file commands, via the temporary file mechanism. This takes advantage of the compiler's ability to open a file and compile the contents by making a temporary file of the marked text. Two variables of interest are:

`sml-temp-threshold`

Default: '`0`'.

Determines what constitutes a large piece of program text. A value of 512, say, will declare half a kilobyte a suitable threshold and larger pieces of text will be sent via the temporary file. A value of 0 means all text is sent via the temporary file; the value nil inhibits the temporary file mechanism altogether.

`sml-temp-file`

Default: '`(make-temp-name "/tmp/ml")`'.

A string that gives the name of the temporary file to use. This default ensures Emacs will invent a unique name for this purpose for use throughout the rest of the editing session. Only one temporary file is used.

Another reason for using the temporary file mechanism is that error messages reported by the ML compiler will generally not make much sense unless a real file is associated with the input (an embedded use file will count as a real file). Of course, this depends on the compiler.

3.4.1 Compiler Settings

The process interaction code in SML mode is independent of the compiler used, so there are a number of variables that need to be set correctly for this to work. Things are by default set up for Standard ML of New Jersey, but switching to a new system is quite easy—you may only need to set the default value of `sml-program-name`. The following SML mode variables need checking:

`sml-default-arg`

Default: '`""`'.

Useful for Poly/ML users, and others who have wrappers for setting various options around the command to run the compiler.

`sml-use-command`

Default: '`"use \"%s\""`'.

Use file command template. Emacs will replace the '`%s`' with a file name. Note that the double quote characters inside the string need quoting with the backslash.

`sml-cd-command`

Default: '`"System.Directory.cd \"%s\""`'.

Compiler command to change the working directory. Perhaps not all ML systems support this feature, but they should.

`sml-prompt-regexp`

Default: '`"^[\-=] *"`'.

What you expect: '`comint`' uses this for various purposes.

A typical way of setting these correctly (other than editing the SML mode source files) is with something like

```
(setq sml-use-command "PolyML.use \"%s\"")
(setq sml-prompt-regexp "^[>#] *")
```

in your '`.emacs`' file (but see Chapter 4 [Configuration], page 12).

3.4.2 Parsing Error Messages

To customise error reportage for different ML compilers you need to set two variables:

`sml-error-regexp`

Default: '`"^.+:[0-9]+\\.[0-9]+.+\\(Error\\|Warning\\):"`'

This is the regular expression for matching an error message. The default matches the Standard ML of New Jersey compiler error and warning messages.

`sml-error-parser`

Default: '`'sml-smlnj-error-parser`'.

The function that actually parses the error message. This should return a list of 3 or 5 elements, viz: source file name, start line, and start column (and optionally, end line and column). Again, the default is for SML/NJ.

See files '`sml-smlnj.el`' and '`sml-poly-ml.el`' to see what (little) needs to be changed for other ML systems.

4 Configuration Summary

This section gives more information on how to reconfigure SML mode, without repeating what has been said before (see Section 2.4 [SML Mode Defaults], page 6, and see Section 3.4 [Process Defaults], page 10). Menus, key bindings, mode hooks and highlighting are described below.

First, the two auxiliary files, '`sml-smlnj.el`' and '`sml-poly-ml.el`' define defaults for these popular (?) ML systems—SML/NJ and Poly/ML.

`sml-smlnj-error-regexp`
`sml-poly-ml-error-regexp`
: (Variable) The pattern matching error and warning messages—should be bound to `sml-error-regexp`.

`sml-smlnj-error-parser`
`sml-poly-ml-error-parser`
: (Function) The function to parse error messages—should be bound to `sml-error-parser`.

`sml-smlnj`
`sml-poly-ml`
: (Function) Set the interaction mode defaults and launch the respective compiler.

The latter functions are available from the SML mode menu; otherwise set them up for autoloading just like '`sml-mode`'—see Section 1.2 [Getting Started], page 2.

4.1 Hooks

One way to set SML mode variables (see Section 2.4 [SML Mode Defaults], page 6), and other defaults, is through the `sml-mode-hook` which you can create in your '`.emacs`'. A simple example:

```
(setq sml-mode-hook
      '(lambda() "ML mode defaults"
         (setq sml-indent-level 2         ; conserve on horizontal space
               words-include-escape t     ; \ loses word break status
               indent-tabs-mode nil)))    ; never indent with tabs
```

The hook is run every time an SML mode buffer is created. ML programmers will recognise the anonymous 'lambda' function. In this case it is not really necessary to set `sml-indent-level` in a hook because this variable is global (all SML mode variables are). With similar effect:

```
(setq sml-indent-level 2)
```

somewhere in your '`.emacs`' file. The variable `indent-tabs-mode` is automatically made local to the current buffer (whenever set) and so must be set in a hook if you always want SML mode to behave like this.

Another hook is `inferior-sml-mode-hook`. This can be used to control the behaviour of the interaction buffer through various variables meaningful to '`comint`'-based packages:

```
(setq inferior-sml-mode-hook
      '(lambda() "Interaction ML mode defaults"
         (setq comint-scroll-show-maximum-output t
               comint-input-autoexpand nil)))
```

Unless you habitually run several ML compilers simultaneously under one Emacs session this hook will normally only get run once.

Note that `sml-load-hook` is run when SML mode is loaded into Emacs, and `sml-mode-hook` is run each time an SML buffer is created; `inferior-sml-mode-hook` is run just when a new ML process is created, after `comint-mode-hook`.

4.2 Key Binding

Customisation (in Emacs) usually entails putting favourite commands on easily remembered keys. Two 'key maps' are defined in SML mode: one is effective in program text buffers (`sml-mode-map`) and the other is effective in the interaction buffer (`inferior-sml-mode-map`). SML mode is set up so that the default key bindings from the former will also be available in the latter.

Type `C-h m` in an SML mode buffer to find the default key bindings (and similarly in an ML interaction buffer). Use the given hooks to install your preferred key bindings:

```
(setq sml-mode-hook
      '(lambda() "SML mode defaults."
         (define-key          sml-mode-map "\C-cd" 'sml-cd)
         (define-key          sml-mode-map "\C-c\C-f" 'sml-send-function)))
```

This has the effect of binding `sml-cd` to the key `C-c d`, and the command `sml-send-function` to the key `C-c C-f`. If you want the same behaviour from `C-c d` in the interaction buffer:

```
(setq inferior-sml-mode-hook
      '(lambda() "Interaction ML mode defaults."
         (define-key inferior-sml-mode-map "\C-cd" 'sml-cd)
         ;; NB. for SML/NJ post v.93
         (setq sml-cd-command "Posix.FileSys.chdir \"%s\"")))
```

4.3 Using Menus and Highlighting

Menus are useful for fiddling with mode defaults and finding out what keys commands are on if you are forgetful (though not all commands are listed in the menu). Highlighting is very handy for picking out keywords in the program text, spotting misspelled keywords, and, if you have '`ps-print`' installed, obtaining prettified code listings.

4.3.1 Menus

For menus to appear in the menu bar under GNU Emacs or GNU XEmacs, the editor must be able to find one of two packages—i.e., one or both must be on you `load-path`. The first option is '`easymenu`' which is distributed with GNU Emacs. Easy!

The second option is '`auc-menu`' which is written by Per Abrahamsen and distributed with AUCTeX, but it is independently available from the IESD lisp archive[2] at Aalborg. You'll also find '`auc-menu`' is available from the LCD archive[3], the main repository for all Emacs lisp. The advantage of '`auc-menu`' is that it works with XEmacs too.

Notice that certain menu entries are not illuminated at first—these are functions that depend on there being an ML process running with which to communicate.

4.3.2 Highlights

Various highlight (hilite, if you want to spell it that way) packages are available for GNU Emacs 19, and GNU XEmacs. SML mode can use either '`hilit19`' which comes with Emacs, or '`font-lock`' which is the package of choice with XEmacs. If you are not familiar with Emacs' highlight packages you'll have to check their sources for installation guidelines, etc..

Use `sml-load-hook` to tell Emacs which scheme you prefer for SML mode. For example:

```
(setq sml-load-hook
      '(lambda() "Highlights." (require 'sml-font)))
```

This ensures the SML extensions to '`font-lock`' will be available once SML mode loads (from '`sml-font.el`'—if you prefer the '`hilit19`' package you should '`(require 'sml-hilite)`' instead.

[2] ftp://ftp.iesd.auc.dk/pub/emacs-lisp/
[3] ftp://archive.cis.ohio-state.edu/pub/gnu/emacs/elisp-archive/

The variable `sml-font-lock-extra-keywords` is for further customising '`font-lock`' for SML mode. The value of the variable should be a list of strings, each of which is a regular expression that should match the desired keyword exactly. Here's an example:

```
(setq sml-font-lock-extra-keywords
      '("\\babstraction\\b" "\\bfunsig\\b" "=>" "::"))))
```

The `\b` marks a word boundary, according to the syntax table defined for SML mode. Backslash must be quoted inside a string. See section "Regexps" in The Emacs Editor Manual, for a summary of Emacs' regular expression syntax. The variable `sml-font-lock-auto-on` can be used to control whether '`font-lock`' should be enabled by default in SML mode buffers, or not.

The `sml-hilite` package is not currently customisable.

5 Credit & Blame

SML Mode was written originally by Lars Bo Nielsen for Emacs 18.5n.

Later hacked for comint by Olin Shivers (who called it ml-mode).

Much later hacked by Matthew Morley because it didn't seem to work...

Fritz Knabe posted the hilites and font-lock functions on the net.

So now there are menus, syntax highlighting, ML compiler independence, TEXinfo, and some hope it'll work with a variety of Emacsen. But there are still things to do. Lars wrote:

- Find a better way to send regions to the inferior shell.
- The indentation algorithm still can be fooled. I don't know if it will ever be 100% right, as this means it will have to actually parse all of the buffer up to the actual line (this can get -very- slow).
- Add tags (rather, a means of generating tags for SML).

One could further add that the correct recognition of functions is also difficult without re-parsing the whole buffer, but someone may have a nice algorithm for this. Advantage could be had from using the multiple frame concept in GNU Emacs 19 and derivatives.

Command Index

I

S

Variable Index

I

S

Table of Contents