An Emacs/Elisp Major Mode for FunnelWeb

A.B.Coates
Department of Physics
The University of Queensland
QLD 4072
Australia
Email: coates@physics.uq.oz.au

June 11, 2010

Contents

Chapter 1

Major and Minor Modes

1.1 The FunnelWeb Major Mode

This is the definition of the function which defines the FunnelWeb mode. A brief (and not exhaustive) description of the functions available is given in its description string.

fw-mode Function Definitions[1] + ≡

```
{
 (defun fw-mode ()
   "Major mode for editing FunnelWeb files.  Built on top of
 tex-mode.  Makes @} , @) , and @> display their matching
 opening braces @{ , @( , or @< .

 Use \\[fw-buffer] to run FunnelWeb on the current buffer
 *without* saving it.
 Use \\[fw-file] to be prompted to save the buffer to a file
 first, before running FunnelWeb on the file.
 Use \\[fw-tex] to run the (La)TeX output file through TeX or
 LaTeX as appropriate.
 Use \\[fw-print] to print a .dvi file.
 Use \\[fw-show-print-queue] to show the print queue that
 \\[fw-print] put your job on.
 Commands from tex-mode (or a similar selected mode) are
 mapped to operate on the (La)TeX file produced by FunnelWeb.

 Key sequences:

 see documentation for 'switch-mode' for key sequences
 relating to mode switching.
```

```
Mode variables:
fw-autofill-mode
        Whether autofill-mode is automatically invoked for
        FunnelWeb files.  Set to 0 for no, to a positive
        number for yes.
fw-TeX-mode
        The major (La)TeX mode on which fw-mode is based.
        Typically either 'tex-mode or 'latex-mode
        (or perhaps 'tex-init for Auc-TeX
         or 'html-mode for hypertext ...).
fw-directory
        Directory in which to create temporary files for
        TeX jobs run by \\[fw-buffer] or \\[fw-file].
fw-use-TeX-quote-style
        Set this to nil so the the \" key produces the
        normal \" character.  If set to true, the \" key
        works as for (La)TeX mode, producing either
        `` or ''.
fw-dvi-print-command
        Command string used by \\[fw-print] to print a
        .dvi file.
fw-show-queue-command
        Command string used by \\[fw-show-print-queue] to
        show the print queue that \\[fw-print] put your
        job on.

Entering FunnelWeb-mode calls the value of fw-mode-hook."
  (interactive)
  (funcall fw-TeX-mode)
  (hack-local-variables)
  (setq mode-name "FunnelWeb")
  (setq major-mode 'fw-mode)
  (auto-fill-mode fw-auto-fill-mode)
  (if fw-mode-map (use-local-map fw-mode-map))
  (setq fw-buffer-p t)
  (switch-mode fw-switch-minor-mode-init)
  (run-hooks 'fw-mode-hook))
}
```

This macro is defined in definitions 1 and 9.
This macro is invoked in definition 37.

1.2 The Switch Minor Mode

This is the definition of the function which defines the Switch minor mode. A brief (and not exhaustive) description of the functions available is given in its description string.

switch-mode Function Definitions 2[2] + ≡

```
{
 (defun switch-mode
   ( &optional set-mode-p
               no-check-p
               region-open-string
               region-close-string
               region-boundary-pattern )
   "With no arguments, toggle whether the region mode
 checking is activated or not.  Optional first argument
 SET-MODE-P can be used to definitely switch the region
 mode checking on or off: a positive number for on, off
 otherwise.
     Optional second argument NO-CHECK-P is nil for a
 check of whether the point is in a code region or not
 to be taken immediately.  With any other value, no check
 is done.
     Optional third and fourth arguments REGION-OPEN-STRING
 and REGION-CLOSE-STRING define the boundaries by which a
 code region is recognised.
     Optional fourth argument REGION-BOUNDARY_PATTERN should
 be a pattern equivalent to
     \\(REGION-OPEN-STRING\\|REGION-CLOSE-STRING\\)
 i.e. a pattern which recognises either an opening or closing
 pattern.

 Key sequences:

 \\C-x\\C-a key sequences are used for the \"Switch\" minor
 mode, which allows modes to be swapped while editing a file.

 \\C-x \\C-a t : toggle Switch minor-mode on and off
 \\C-x \\C-a c : check whether in macro region or not
 \\C-x \\C-a p : toggles whether the default macro region
                 mode is used with or without prompting
 \\C-x \\C-a s : set the current macro region mode"
   (interactive)
 }
```

This macro is defined in definitions 2, 4, 6, 14, 15, 16, 18, 19, 20, 21, 23, 26 and 30.
This macro is invoked in definition 32.

A variable, local to each buffer, is needed to note whether switch-mode is active or not.

switch-mode Variable Definitions 1[3] + ≡

```
{
 (defvar switch-mode nil
   "Whether the region checking minor mode is active or not.")
 (make-variable-buffer-local 'switch-mode)
 }
```

This macro is defined in definitions 3, 5, 13, 17 and 29.
This macro is invoked in definition 32.

The boolean value `switch-on-p` is given the value `t` if the switch minor mode should be turned on, `nil` otherwise. The value depends on when the function call toggles the state or sets it directly.

switch-mode Function Definitions 2[4] + ≡

```
{  (let ((switch-on-p
          (or (and (numberp set-mode-p)
                   (> set-mode-p 0))
              (not switch-mode))))
     (if switch-on-p
         (let ((old-switch-mode-value
                switch-mode))
           (setq switch-mode t
                 switch-force-region-check-p t)
           (if (not (or no-check-p old-switch-mode-value))
               (switch-check-if-in-region)))
       (setq switch-mode nil))
 }
```

This macro is defined in definitions 2, 4, 6, 14, 15, 16, 18, 19, 20, 21, 23, 26 and 30.
This macro is invoked in definition 32.

Three variables, with local values in each buffer, are required to hold the values of the region opening and closing strings and a combined pattern.

switch-mode Variable Definitions 1[5] + ≡

```
{
 (defvar switch-open-region-string "@{"
   "String defining the beginning of a code region.")
 (make-variable-buffer-local 'switch-open-region-string)

 (defvar switch-close-region-string "@}"
```

```
  "String defining the end of a code region.")
(make-variable-buffer-local 'switch-close-region-string)

(defvar switch-boundary-region-pattern "@\\({\\|}\\)"
  "Regular expression defining the beginning or end of a code
region.")
(make-variable-buffer-local 'switch-boundary-region-pattern)
}
```

This macro is defined in definitions 3, 5, 13, 17 and 29.
This macro is invoked in definition 32.

switch-mode Function Definitions 2[6] + ≡

```
{    (if region-open-string
         (setq switch-open-region-string
               region-open-string))
     (if region-close-string
         (setq switch-close-region-string
               region-close-string))
     (if region-boundary-pattern
         (setq switch-boundary-region-pattern
               region-boundary-pattern))
     (force-mode-line-update t)))
}
```

This macro is defined in definitions 2, 4, 6, 14, 15, 16, 18, 19, 20, 21, 23, 26 and 30.
This macro is invoked in definition 32.

1.3 Invocation of the FunnelWeb mode

To cause FunnelWeb to be invoked for `.fw` and `.fwi` files, the following lines should be added to the user's "`.emacs`" file.

Emacs initialisation file additions[7]**Z** ≡

```
{
 (autoload 'fw-mode "fw-mode")
 (setq
  auto-mode-alist
  (reverse
   (append
    (reverse auto-mode-alist)
    (list
     (append
      (list "\\.fw$")
      'fw-mode
```

```
        )
       (append
        (list "\\.fwi$")
        'fw-mode
        )
       )
      )
     )
    )
   }
```

This macro is NEVER invoked.

Chapter 2

Function and Variable Definitions for the FunnelWeb Mode

An alternative command for saving buffers converts all tabs to spaces before saving, since FunnelWeb does not accept tab characters in .fw files by default. Because mode-switching is expected to be used when editing FunnelWeb files, a local variable in each buffer is set to define whether a buffer is a FunnelWeb buffer or not. A global variable is also provided to switch off the replacement of tabs by spaces. The string used to replace tabs with spaces can also be changed from the default of 8 spaces. Another global variable also controls the removal of trailing spaces, since these are also not accepted by FunnelWeb by default.

fw-mode Variable Definitions2[8] + ≡

```
{
 (defvar fw-untabify-before-saving-p t
   "*Whether to convert tabs to spaces before saving FunnelWeb
 buffers.")

 (defvar fw-tab-string " "
   "Tab character as string.")

 (defvar fw-tab-space-replacement "        "
   "*String of spaces for replacing tabs.")
 (make-variable-buffer-local 'fw-tab-space-replacement)

 (defvar fw-remove-trailing-spaces-before-saving-p t
```

```
  "*Whether to remove trailing spaces on lines before saving
FunnelWeb buffers.")

(defvar fw-space (string-to-char " ")
  "Space character, used in removing trailing spaces.")

(defvar fw-trailing-space-pattern " +$"
  "*Pattern used to identify trailing spaces on lines.")

(defvar fw-null-string ""
  "Empty string used for replacing trailing spaces.")
}
```

This macro is defined in definitions 8 and 11.
This macro is invoked in definition 36.

fw-mode Function Definitions[9] + ≡

```
{
(defun fw-untabify-save-buffer ()
  "Save the buffer, optionally removing tabs and trailing
spaces
first if the buffer is a FunnelWeb buffer."
  (interactive)
  (if fw-buffer-p
      (progn
        (if fw-untabify-before-saving-p
            (progn
              (message "Removing tabs ...")
              (save-excursion
                (goto-char (point-min))
                (while (search-forward fw-tab-string nil t)
                  (replace-match fw-tab-space-replacement nil
t)))))
        (if fw-remove-trailing-spaces-before-saving-p
            (progn
              (message "Removing trailing spaces ...")
              (save-excursion
                (goto-char (point-min))
                (while (< (point) (point-max))
                  (end-of-line)
                  (while (eq (preceding-char) fw-space)
                    (backward-delete-char 1))
                  (forward-line 1)))))))
  (save-buffer))
}
```

This macro is defined in definitions 1 and 9.
This macro is invoked in definition 37.

Other variable definitions follows. Those for which the decription text begins with a `*` can be changed on the command line by the user, and all can be set in the users `.emacs` file.

fw-mode Variable Definitions1[10] + ≡

```
{
 (defvar fw-TeX-mode 'tex-mode
   "*The default TeX-based foundation for FunnelWeb mode.")
 }
```

This macro is defined in definitions 10.
This macro is invoked in definition 36.

fw-mode Variable Definitions2[11] + ≡

```
{
 (defvar fw-switch-minor-mode-init 1
   "*Whether to activate the Switch minor mode automatically
 on entering FunnelWeb mode.  A positive integer for yes.")

 (defvar fw-buffer-p nil
   "Whether a buffer is a FunnelWeb buffer or not.")
 (make-variable-buffer-local 'fw-buffer-p)
 (switch-add-to-preservation-list 'fw-buffer-p)

 (defvar fw-quote-style t
   "*Whether to map \" to a single character or use (La)TeX
 mapping to `` or '' as appropriate.  Can be set to nil to
 remove the (La)TeX mapping, or anything else to enable it.")

 (defvar fw-auto-fill-mode 1
   "*Whether autofill-mode is automatically invoked for
 FunnelWeb files.  Set to 0 for no, to a positive number
 for yes.")

 (defvar fw-command "fw"
   "*The command to run FunnelWeb on a file.
 Any pre-options (fw-command-pre-options) will be appended
 to this string, separated by a space, followed by the
 filename, also separated by a space, and finally any
 post-options (fw-command-post-options), again separated
 by a space.")

 (defvar fw-command-pre-options nil
```

```
  "*Options which go before the filename when
calling FunnelWeb.")
(make-variable-buffer-local 'fw-command-pre-options)

(defvar fw-command-post-options "+t +D"
  "*Options which go after the filename when
calling FunnelWeb.")
(make-variable-buffer-local 'fw-command-post-options)

(defvar fw-shell-cd-command "cd"
  "*Command to give to shell running FunnelWeb to
change directory.  The value of fw-directory will be
appended to this, separated by a space.")

(defvar fw-mode-syntax-table nil
  "Syntax table used while in FunnelWeb mode.")

(defvar fw-mode-map nil
  "Keymap for FunnelWeb mode.")

(defvar fw-close-definition-block-additive t
  "*Whether definitions should be closed using \"+=\" (t) or
\"==\" (nil).")
(make-variable-buffer-local
 'fw-close-definition-block-additive)

(defvar fw-close-definition-block-newline-suppress t
  "*Whether definitions should be closed using \"@-\".")
(make-variable-buffer-local
 'fw-close-definition-block-newline-suppress)

(defvar fw-close-definition-block-blank-line t
  "Whether definitions should be closed leaving
a blank line.")
(make-variable-buffer-local
 'fw-close-definition-block-blank-line)
}
```

This macro is defined in definitions 8 and 11.
This macro is invoked in definition 36.

Chapter 3

Key Definitions for the FunnelWeb Mode

These are the special key definitions for FunnelWeb commands.

fw-mode Key Definitions[12] ≡

```
{
 (global-set-key "\C-x\C-s" 'fw-untabify-save-buffer)
 }
```

This macro is invoked in definition 38.

Chapter 4

Function and Variable Definitions for the Switch Minor Mode

The Switch minor mode works by adding a function, run after each emacs command, which checks the position of point. If point is outside of the current region, each code or text, a function is run to determine whether point lies in a code or text region. Note that for speed, the code does not update the boundaries of the current region after each command, and so the checking may be fooled when and if sections are deleted from a code or text region.

Variables are needed to hold the bounds of the current region and the whether flag marking whether a region check is in progress or not (to avoid an infinite recursion).

switch-mode Variable Definitions 1[13] + ≡

```
{
 (defvar switch-region-min nil
   "Start of current macro or text region, or nil.")
 (make-variable-buffer-local 'switch-region-min)

 (defvar switch-region-max nil
   "End of current macro or text region, or nil.")
 (make-variable-buffer-local 'switch-region-max)

 (defvar switch-not-in-region-check-routine-p t
   "Whether a region check is not in progress.")
```

```
(make-variable-buffer-local
 'switch-not-in-region-check-routine-p)
}
```

This macro is defined in definitions 3, 5, 13, 17 and 29.
This macro is invoked in definition 32.

switch-mode Function Definitions 2[14] + ≡

```
{
(defun switch-post-command-hook-function ()
  "Function to check if the user is in a code definition
region or not."
  (if (and (not isearch-mode)
           switch-mode
           switch-not-in-region-check-routine-p
           (or (not switch-region-min)
               (not switch-region-max)
               (let ((the-point (point)))
                 (or
                  (< the-point switch-region-min)
                  (> the-point switch-region-max)))))
      (let ((switch-not-in-region-check-routine-p nil))
        (switch-check-if-in-region))))
}
```

This macro is defined in definitions 2, 4, 6, 14, 15, 16, 18, 19, 20, 21, 23, 26 and 30.
This macro is invoked in definition 32.

The function for checking whether the point is in a code or text region or not is as follows.

switch-mode Function Definitions 2[15] + ≡

```
{
(defun switch-check-if-in-region ()
  "Check if point is inside or outside a FunnelWeb macro
definition. If outside, switch to (Auc/La)TeX mode.
If inside, either ask the user for a mode, defaulting
to the last mode used in a macro definition, or directly
use the last mode chosen (depends on value of variable
switch-mode-prompt-p)."
  (interactive)
}
```

This macro is defined in definitions 2, 4, 6, 14, 15, 16, 18, 19, 20, 21, 23, 26 and 30.
This macro is invoked in definition 32.

First, searches are carried out forwards and backwards from point to determine what sort of region point is in.

switch-mode Function Definitions 2[16] + ≡

```
{  (let ((backward-search-result
          (switch-search-backward-check
           switch-boundary-region-pattern
           switch-open-region-string))
          (forward-search-result
          (switch-search-forward-check
           switch-boundary-region-pattern
           switch-close-region-string))
          (backward-search-value)
          (forward-search-value))
      (setq backward-search-value
            (car backward-search-result))
      (setq forward-search-value
            (car forward-search-result))
      (setq switch-region-min
            (car (cdr backward-search-result)))
      (setq switch-region-max
            (car (cdr forward-search-result)))
      (if (not (or (< backward-search-value 0)
                   (< forward-search-value 0)
                   (and (eq backward-search-value 0)
                        (eq forward-search-value 0))))
 }
```

This macro is defined in definitions 2, 4, 6, 14, 15, 16, 18, 19, 20, 21, 23, 26 and 30.
This macro is invoked in definition 32.

The following code determines what is done if point is inside a code region. Variables are required to define the behaviour, and to store the current code region mode, as well as to store whether point was last in a code or text region.

switch-mode Variable Definitions 1[17] + ≡

```
{
 (defvar switch-force-region-check-p nil
   "Whether a region check should be forced or not,
 regardless of whether it seems to be required or not.")
 (make-variable-buffer-local 'switch-force-region-check-p)

 (defvar switch-mode-prompt-p t
   "*Whether the user should be prompted for a new
 major mode type each time point enters a code region.")
 (make-variable-buffer-local 'switch-mode-prompt-p)

 (defvar switch-current-mode nil
   "The current mode for a code region.")
```

```
(make-variable-buffer-local 'switch-current-mode)

(defvar switch-currently-in-region-p nil
  "Whether point is in a code region or not.")
(make-variable-buffer-local 'switch-currently-in-region-p)

(defvar switch-highlight-p nil
  "Whether to highlight regions as they are entered.
(This requires the `hilit19' library to be loaded.)")
(make-variable-buffer-local 'switch-highlight-p)
}
```

This macro is defined in definitions 3, 5, 13, 17 and 29.
This macro is invoked in definition 32.

switch-mode Function Definitions 2[18] + ≡

```
{       (if (or (not switch-currently-in-region-p)
                switch-force-region-check-p)
            (progn
              (switch-select-mode (not switch-mode-prompt-p))
              (if switch-current-mode
                  (progn
                    (message (concat "Changed to "
                                     (prin1-to-string
                                      switch-current-mode)))
                    (setq switch-currently-in-region-p t)))))
}
```

This macro is defined in definitions 2, 4, 6, 14, 15, 16, 18, 19, 20, 21, 23, 26 and 30.
This macro is invoked in definition 32.

Now the code for the case where point is *not* in a code region, but in a text region.

switch-mode Function Definitions 2[19] + ≡

```
{     (if switch-currently-in-region-p
          (let ((preservation-list
                 (switch-get-preservation-values)))
            (fw-mode)
            (switch-set-preservation-values
             preservation-list)
            (message "Changed to fw-mode")
            (setq switch-currently-in-region-p nil))))
      (setq switch-force-region-check-p nil)))
}
```

This macro is defined in definitions 2, 4, 6, 14, 15, 16, 18, 19, 20, 21, 23, 26 and 30.
This macro is invoked in definition 32.

A function is provided to allow the user to toggle whether the mode is prompted for each time a code region is entered or not.

switch-mode Function Definitions 2[20] + ≡

```
{
 (defun switch-mode-prompt-toggle ()
   "Toggle whether the user is prompted for the major mode
 each time a code region is entered."
   (interactive)
   (setq switch-mode-prompt-p (not switch-mode-prompt-p))
   (if switch-mode-prompt-p
       (message "Mode prompting switched on.")
     (message "Mode prompting switched off.")))
}
```

This macro is defined in definitions 2, 4, 6, 14, 15, 16, 18, 19, 20, 21, 23, 26 and 30.
This macro is invoked in definition 32.

It is also sometimes useful to be able to force a major mode to be asked for. This is necessary when editing changes cause the program to mistake the boundaries of the current region.

switch-mode Function Definitions 2[21] + ≡

```
{
 (defun switch-mode-force-prompt ()
   "Cause the user to be prompted for a mode type for
 the current region, if a code region."
   (interactive)
   (let ((switch-force-region-check-p t))
     (switch-check-if-in-region)))
}
```

This macro is defined in definitions 2, 4, 6, 14, 15, 16, 18, 19, 20, 21, 23, 26 and 30.
This macro is invoked in definition 32.

The mode selection function allows the user to choose from any command which ends with string "-mode". It optionally beeps on activation, to note to the user that a mode selection is necessary.

switch-mode Variable Definitions 2[22] + ≡

```
{
 (defvar switch-mode-list (make-switch-mode-list)
   "Return a list of strings of all possible major modes
 from which the user can choose.")

 (defvar switch-beep-on-mode-selection-p t
   "*Whether to beep when a mode selection is necessary.")
}
```

This macro is defined in definitions 22.
This macro is invoked in definition 32.

switch-mode Function Definitions 2[23] + ≡

```
{
 (defun switch-select-mode ( &optional no-prompt-p )
   "Interactively get the user to select a macro mode,
 giving the last-used macro mode as a default, and allowing
 the user to select from all possible major mode commands
 with name completion.
     Optional parameter NO-PROMPT-P, if non-nil, stops
 prompting from taking place unless the current mode
 (switch-current-mode) is nil."
   (let ((old-switch-mode-value
          (if switch-mode 1 0)))
     (if (not (and no-prompt-p switch-current-mode))
         (progn
           (if switch-beep-on-mode-selection-p (beep))
           (let ((new-mode (completing-read
                            "Mode: "
                            switch-mode-list
                            nil
                            nil
                            (switch-convert-to-default-string
                             switch-current-mode)
                            nil)))
             (setq switch-current-mode
                   (car (read-from-string new-mode))))))
     (let ((preservation-list
            (switch-get-preservation-values))
           (region-min switch-region-min)
           (region-max switch-region-max))
       (message "Changing to %s"
                (prin1-to-string switch-current-mode))
       (if switch-current-mode (funcall switch-current-mode))
       (if (and switch-highlight-p region-min region-max)
           (progn
             (hilit-unhighlight-region region-min region-max)
             (hilit-highlight-region region-min region-max)))
       (switch-set-preservation-values preservation-list))
     (switch-mode old-switch-mode-value t)
     (force-mode-line-update)))
}
```

This macro is defined in definitions 2, 4, 6, 14, 15, 16, 18, 19, 20, 21, 23, 26 and 30.

This macro is invoked in definition 32.

A short function to convert an object to a string if non-nil.

switch-mode Function Definitions 1[24] + ≡

```
{
 (defun switch-convert-to-default-string ( object )
   "Convert an object into a default string for
 name completion."
   (if object (prin1-to-string object)))
 }
```

This macro is defined in definitions 24, 25, 27 and 28.
This macro is invoked in definition 32.

The following function returns a list of possible modes, in the form of an Emacs "alist". A second function allows the user to update the list.

switch-mode Function Definitions 1[25] + ≡

```
{
 (defun make-switch-mode-list ()
   "Return a list of all commands ending in \"-mode\",
 in the form of an alist."
   (let ((mode-list (apropos-internal "-mode$")))
     (mapcar 'switch-convert-to-alist-string mode-list)))
 }
```

This macro is defined in definitions 24, 25, 27 and 28.
This macro is invoked in definition 32.

switch-mode Function Definitions 2[26] + ≡

```
{
 (defun update-switch-mode-list ()
   "Update the mode list used for the switch minor-mode."
   (interactive)
   (setq switch-mode-list (make-switch-mode-list)))
 }
```

This macro is defined in definitions 2, 4, 6, 14, 15, 16, 18, 19, 20, 21, 23, 26 and 30.
This macro is invoked in definition 32.

This simple function converts objects to a form suitable for an Elisp "alist".

switch-mode Function Definitions 1[27] + ≡

```
{
 (defun switch-convert-to-alist-string ( object )
   "Convert an object to a string in its own list,
 the format appropriate for an alist."
   (list (prin1-to-string object)))
```

```
}
```

This macro is defined in definitions 24, 25, 27 and 28.
This macro is invoked in definition 32.

Two functions are used to search for code region boundaries, one for searching forwards from point, the other for searching backwards. Each searches for the given regexp, and if found, compares it to the given string. A list is returned containing an integer value describing the search result and the position of the string which was matched.

switch-mode Function Definitions 1[28] + ≡

```
{
(defun switch-search-forward-check ( regexp string )
  "Search forward for the given regexp, and if the matching
pattern is found and it matches the given string, return `1'
and the beginning of the pattern, else return `-1' and the
beginning of the pattern.  If the pattern is not found when
the end of the buffer is reached, return `0' and the end of
the buffer."
  (save-excursion
    (if (search-forward-regexp regexp (point-max) t)
        (if (string-equal
             string
             (buffer-substring
              (match-beginning 0)
              (match-end 0)))
            (list 1 (match-beginning 0))
          (list -1 (match-beginning 0)))
      (list 0 (point-max)))))

(defun switch-search-backward-check ( regexp string )
  "Search backward for the given regexp, and if the matching
pattern is found and it matches the given string, return `1'
and the end of the pattern, else return `-1' and the end of
the pattern.  If the pattern is not found when the beginning
of the buffer is reached, return `0' and the beginning of the
buffer."
  (save-excursion
    (if (search-backward-regexp regexp (point-min) t)
        (if (string-equal
             string
             (buffer-substring
              (match-beginning 0)
              (match-end 0)))
            (list 1 (match-end 0))
```

```
            (list -1 (match-end 0)))
          (list 0 (point-min)))))
    }
```

This macro is defined in definitions 24, 25, 27 and 28.
This macro is invoked in definition 32.

It is useful to allow functions to maintain a list of variables which should be preserved by switch-mode operations. This list can be changed by the user.

switch-mode Variable Definitions 1[29] + ≡

```
    {
     (defvar switch-preservation-list
       (list 'switch-current-mode 'switch-mode-prompt-p)
       "List of names of variables that need to be specially
     preserved by switch-mode operations.")
     }
```

This macro is defined in definitions 3, 5, 13, 17 and 29.
This macro is invoked in definition 32.

switch-mode Function Definitions 2[30] + ≡

```
    {
     (defun switch-add-to-preservation-list ( var )
       "Add a variable to the switch-mode preservation list."
       (if (not (memq var switch-preservation-list))
           (setq switch-preservation-list
                 (append switch-preservation-list (list var)))))

     (defun switch-get-preservation-values ()
       "Returns a list of current values which can be used by
     switch-set-preservation-values to restore the values of
     the variables in the preservation list
     (switch-preservation-list)."
       (if switch-preservation-list
           (mapcar
            'switch-preserve-form
            switch-preservation-list)))

     (defun switch-preserve-form ( var )
       "Return a list of the variable name and its value."
       (list var (eval var)))

     (defun switch-set-preservation-values ( vallist )
       "Reset the variables in the preservation value list
     using the value list previously created by
     switch-get-preservation-values."
```

```
  (mapcar 'switch-apply-set vallist))

(defun switch-apply-set ( varpair )
  "Given a list of the form (NAME . VALUE), assign the value
to the name."
  (apply 'set varpair))
}
```

This macro is defined in definitions 2, 4, 6, 14, 15, 16, 18, 19, 20, 21, 23, 26 and 30.
This macro is invoked in definition 32.

Chapter 5

Key Definitions for the Switch Minor Mode

These are the special key definitions for Switch minor-mode commands.

switch-mode Key Definitions[31] ≡

```
{
 (defvar switch-prefix nil
   "Switch minor-mode region check \\C-x\\C-a keymap.")

 (define-prefix-command 'switch-prefix)

 (global-set-key "\C-x\C-a" 'switch-prefix)

 (define-key switch-prefix "c" 'switch-check-if-in-region)

 (define-key switch-prefix "t" 'switch-mode)

 (define-key switch-prefix "p" 'switch-mode-prompt-toggle)

 (define-key switch-prefix "s" 'switch-mode-force-prompt)
 }
```

This macro is invoked in definition 32.

Chapter 6

Construction of the elisp files

The file "`switch-mode.el`" requires the standard Emacs distribution provides the new minor mode function '`switch-mode`'.

The variable and function definitions are each divided up into two lots, in the order

- variables which don't rely on functions for their default values
- functions which don't rely on variables in their definitions
- variables which rely on functions for their default values
- functions which rely on variables in their definitions.

switch-mode Constructed Code[32] + ≡
{`(provide 'switch-mode)`

switch-mode Variable Definitions 1[3]

switch-mode Function Definitions 1[24]

switch-mode Variable Definitions 2[22]

switch-mode Function Definitions 2[2]

switch-mode Key Definitions[31]

```
}
```
This macro is defined in definitions 32 and 33.
This macro is invoked in definition 35.

The following code installs the minor mode and runs any user-provided hook function.

switch-mode Constructed Code[33] + ≡
```
{(setq minor-mode-alist
       (reverse
        (append (reverse minor-mode-alist)
                (list (list
                       'switch-mode
                       " Switch")))))
 (force-mode-line-update t)

 (add-hook 'post-command-hook
           'switch-post-command-hook-function)

 (run-hooks 'switch-mode-hook)}
```
This macro is defined in definitions 32 and 33.
This macro is invoked in definition 35.

Standard Header[34]**M** ≡
```
{; A.B.Coates
 ; Department of Physics
 ; The University of Queensland  QLD  4072
 ; Australia
 ; Standard GNU copyleft provisions apply to this file.}
```
This macro is invoked in definitions 35 and 39.

switch-mode.el[35] ≡
```
{
 Standard Header[34]

 switch-mode Constructed Code[32]
 }
```
This macro is attached to an output file.

The file "`fw-mode.el`" requires the switch minor mode file "`switch-mode.el`", and provides the new mode function '`fw-mode`'. Calling the selected TEX-mode is a way of making sure that the required definitions are loaded, since the mode-selection function may be defined as an *autoload* function.

fw-mode Constructed Code[36] + ≡
```
{(require 'switch-mode)
```

```
(provide 'fw-mode)
```

fw-mode Variable Definitions1[10]

```
(funcall fw-TeX-mode)
```

fw-mode Variable Definitions2[8]
}

This macro is defined in definitions 36, 37 and 38.
This macro is invoked in definition 39.

In case the user forgets the precise name, some synonyms for '`fw-mode`' are provided.

fw-mode Constructed Code[37] + ≡

```
{(fset 'FW-mode 'fw-mode)
 (fset 'funnelweb-mode 'fw-mode)
 (fset 'FunnelWeb-mode 'fw-mode)
```

fw-mode Function Definitions[1]}

This macro is defined in definitions 36, 37 and 38.
This macro is invoked in definition 39.

If the keymap has not already been set in the user's "`.emacs`" file, then a TeX-mode keymap is copied and used if possible.

fw-mode Constructed Code[38] + ≡

```
{(if fw-mode-map
  (progn
   (defvar tex-mode-map nil)
   (if tex-mode-map
       (setq fw-mode-map tex-mode-map)
     (progn
       (defvar latex-mode-map nil)
       (if latex-mode-map
           (setq fw-mode-map latex-mode-map)
         (progn
           (defvar LaTeX-mode-map nil)
           (if LaTeX-mode-map
               (setq fw-mode-map LaTeX-mode-map)
             (progn
               (defvar TeX-mode-map nil)
               (if TeX-mode-map
                   (setq fw-mode-map TeX-mode-map)))))))))))
```

fw-mode Key Definitions[12]}

This macro is defined in definitions 36, 37 and 38.
This macro is invoked in definition 39.

fw-mode.el[39] ≡
{
Standard Header[34]

fw-mode Constructed Code[36]
}

This macro is attached to an output file.