```
;;; tidy.el --- Interface to the HTML Tidy program

;; Copyright (C) 2001, 2002 by Free Software Foundation, Inc.

;; Time-stamp: <2002-07-09 14:47:02 kahlil>

;; Emacs Lisp Archive Entry
;; Filename: tidy.el
;; Author: Kahlil (Kal) HODGSON <dorge@tpg.com.au>
;; Version: 2.8

;; Keywords: languages
;; X-URL: http://csl.anu.edu.au/~kahlil/emacs/

;; This file is free software; you can redistribute it and/or modify
;; it under the terms of the GNU General Public License as published by
;; the Free Software Foundation; either version 2, or (at your option)
;; any later version.

;; This file is distributed in the hope that it will be useful,
;; but WITHOUT ANY WARRANTY; without even the implied warranty of
;; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
;; GNU General Public License for more details.

;; You should have received a copy of the GNU General Public License
;; along with GNU Emacs; see the file COPYING.  If not, write to
;; the Free Software Foundation, Inc., 59 Temple Place - Suite 330,
;; Boston, MA 02111-1307, USA.

;;;; Commentary:

;; Provides a simple interface to the HTML Tidy program -- a free
;; utility that can fix common errors in your mark-up and clean up
;; sloppy editing automatically. See
;;
;;       <http://tidy.sourceforge.net/>
;;
;; for more details.  This package provides the following functions:
;;
;;       `tidy-buffer',
;;       `tidy-parse-config-file',
;;       `tidy-save-settings', and
;;       `tidy-describe-options',
;;
;; These can be invoked interactively (using M-x) or via the menu-bar.
;; The function `tidy-buffer' sends the current buffer to HTML Tidy,
;; replacing the existing contents with a "tidied" version.  If
;; `tidy-buffer' is given a prefix argument, tidy operates on the
;; current region, ignoring mark-up outside <BODY>...</BODY> tags
;; (useful for writhing cgi scripts in Pearl).  Warnings and errors
;; are presented in a compilation buffer to facilitate tracking down
;; necessary changes (e.g. C-x ` is bound to `next-error').
;;
;; This package also provides menu-bar support for setting Tidy's many
;; options, and includes support for Tidy configuration files.  The
;; function `tidy-parse-config-file' will synchronise options
;; displayed in the menu-bar with the settings in `tidy-config-file'.
;; This is normally called by the load-hook for your HTML editing mode
;; (see installation instructions below).  The function
;; `tidy-save-settings' will save the current option settings to your
;; `tidy-config-file'.  Finally `tidy-describe-options' allows you to
;; browse the documentation strings associated with each option.

;;;
```

```
;;;; Installation:

;; This package assumes you have and up-to-date HTML Tidy program
;; installed on your system.  See the URL above for instructions on
;; how to do this.  To set up this support package, first place the
;; "tidy.el" file somewhere in your `load-path' and open it in Emacs.
;; Byte-compile and load this package using the command
;;
;; M-x emacs-lisp-byte-compile-and-load <RET>
;;
;; Next customise the variables `tidy-config-file',
;; `tidy-shell-command', `tidy-menu-lock' and `tidy-menu-x-position'
;;
;; M-x customize-group <RET> tidy <RET>
;;
;; Now add the following autoloads to your ".emacs.el" file:
;;
;; (autoload 'tidy-buffer "tidy" "Run Tidy HTML parser on current buffer" t)
;; (autoload 'tidy-parse-config-file "tidy" "Parse the `tidy-config-file'" t)
;; (autoload 'tidy-save-settings "tidy" "Save settings to `tidy-config-file'" t)
;; (autoload 'tidy-build-menu  "tidy" "Install an options menu for HTML Tidy." t)
;;
;; If you use html-mode to edit HTML files then add something like
;; this as well

;; (defun my-html-mode-hook () "Customize my html-mode."
;;   (tidy-build-menu html-mode-map)
;;   (local-set-key [(control c) (control c)] 'tidy-buffer)
;;   (setq sgml-validate-command "tidy"))
;;
;; (add-hook 'html-mode-hook 'my-html-mode-hook)

;; This will set up a "tidy" menu in the menu bar and bind the key
;; sequence "C-c C-c" to `tidy-buffer' in html-mode (normally bound to
;; `validate-buffer').
;;
;; For other modes (like html-helper-mode) simple change the variables
;; `html-mode-hook' and `html-mode-map' to whatever is appropriate e.g.

;; (defun my-html-mode-hook () "Customize my html-helper-mode."
;;   (tidy-build-menu html-helper-mode-map)
;;   (local-set-key [(control c) (control c)] 'tidy-buffer)
;;   (setq sgml-validate-command "tidy"))
;;
;; (add-hook 'html-helper-mode-hook 'my-html-mode-hook)

;; Finally, restart Emacs and open an HTML file to test-drive the tidy
;; package. For people new to HTML tidy check that the option "markup"
;; under the "Input/Output" sub menu is set. You can read the
;; documentation on this option via the menu item "Describe Options".
;;
;; Enjoy!

;;;; New Features:
;;
;; 1. Improved menu support to facillitate incorporting new options
;; 2. Menu lock option makes menu stick when toggling options.
;; 3. Now runs on XEmacs!!
;; 4. Uses error file rather than std-error to retrieve errors (this
;;    fixes some odd pop up behaviour)
;; 5. minor bug fix (empty config files)
;; 6. handle buffer modified query in error buffer better
;; 7. make it impossible to mark the error buffer as modified
```

```
;;;; ToDo:
;;
;; 1. Automatically set "char-encoding" according to the buffer encoding

;;;; Bugs:

;; Requires a version of HTML Tidy that understands the "--error-file"
;; "-config" "--show-body-only" command line options e.g. source-forge
;; pre-release.
;;
;; There may be a bug with setting doctypes.  I don't use this feature
;; yet and, well, don't really know how its supposed to work:-)
;;
;; Care with character encodings!!

;;;; Credits

;; This code was inspired by an Emacs "tip" suggested by Pete Gelbman.
;;
;; Thanks to Hans-Michael Stahl for comments regarding XEmacs
;; compatibility.
;;
;; Thanks to Thomas Baumann for bugfix's in `tidy-parse-config-file'
;; and `tidy-buffer'.
;;
;; Thanks to Chris Lott for comments regarding installation and menu
;; display

;;;; Code:

;;;;; Forward references (stuff which must come first)

(require 'easymenu) ;; This makes menus so much easier!
(require 'compile)  ;; To make the error buffer more sexy

;; The following two are functions so that the same compiled code will
;; work in both situations (time cost is negligible)

(defsubst tidy-xemacs-p ()
  "Return t iff we are running XEmacs this session."
  (not (null (string-match "^XEmacs.*" (emacs-version)))))

(defsubst tidy-windows-p ()
  "Return t iff we are running on a Windows system."
  (memq system-type '(emx win32 w32 mswindows ms-dos windows-nt)))

;; function definitions

;; XEmacs
(defalias 'tidy-x-event-function          'event-function)
(defalias 'tidy-x-event-object            'event-object)
(defalias 'tidy-x-find-menu-item          'find-menu-item)
(defalias 'tidy-x-get-popup-menu-response 'get-popup-menu-response)
(defalias 'tidy-x-make-event              'make-event)
(defalias 'tidy-x-misc-user-event-p       'misc-user-event-p)

;;;;; User Variables

(defgroup tidy nil
"*Provides a simple interface to the HTML Tidy program -- a free
utility that can fix common errors in your mark-up and clean up
sloppy editing automatically. See

      <http://tidy.sourceforge.net/>
```

```
for more details.  This package provides the following functions:

      `tidy-buffer',
      `tidy-parse-config-file',
      `tidy-save-settings', and
      `tidy-describe-options',

These can be invoked interactively (using M-x) or via the menu-bar.
The function `tidy-buffer' sends the current buffer to HTML Tidy,
replacing the existing contents with a \"tidied\" version.  If
`tidy-buffer' is given a prefix argument, tidy operates on the
current region, ignoring mark-up outside <BODY>...</BODY> tags
\(useful for writhing cgi scripts in Pearl).  Warnings and errors
are presented in a compilation buffer to facilitate tracking down
necessary changes (e.g. C-x ` is bound to `next-error').

This package also provides menu-bar support for setting Tidy's many
options, and includes support for Tidy configuration files.  The
function `tidy-parse-config-file' will synchronise options
displayed in the menu-bar with the settings in `tidy-config-file'.
This is normally called by the load-hook for your HTML editing mode
\(see installation instructions below).  The function
`tidy-save-settings' will save the current option settings to your
`tidy-config-file'.  Finally `tidy-describe-options' allows you to
browse the documentation strings associated with each option.
")


(defcustom tidy-config-file "~/.tidyrc"
  "*Path to your default tidy configuration file.

This is used by `tidy-parse-config-file' to synchronise Tidy's behaviour
inside Emacs with its behaviour outside, and by `tidy-save-settings' to
set your configuration file from within Emacs.  If you never want this to
happen, set `tidy-config-file' to nil."
  :group 'tidy
  :type 'string)

(defcustom tidy-shell-command "/usr/local/bin/tidy"
  "*Full path to command to call HTML tidy from a shell."
  :group 'tidy
  :type 'string)

(defcustom tidy-menu-lock t
  " *Non-nil means menu is locked (i.e. doesn't pop down) when
selecting toggle and radio options.

See also `tidy-menu-x-position'."
  :type 'boolean
  :group 'tidy)

(defcustom tidy-menu-x-position 211
  "*Specify menu position height in pixels.

This variable is used to set the horizontal position of the locked
menu, so don't forget to adjust it if menu position is not ok.

See also `tidy-menu-lock'."
  :type 'integer
  :group 'tidy)

;;;;; Local Variables

(defvar tidy-debug  nil
  "If t then we rebuild everything on reload. Useful for debugging.")
```

```
(eval-when-compile (setq tidy-debug t))

(defun tidy-toggle-debug () "Toggle value of tidy-debug."
  (interactive)
  (message "tidy-debug is %s" (setq tidy-debug (not tidy-debug))))

(defvar tidy-options-alist nil
  "An alist containing all valid tidy options.
Each element is a list of the form
    (NAME, SUB-MENU, VALUE-TYPE, DEFAULT-VALUE, DOC-STRING).
This is used to automatically construct variables and a menu bar.
To add new or modify exiting options simply modify this list.")

(when (or (null tidy-options-alist) tidy-debug)
  (setq tidy-options-alist
        '(
          ("add-xml-decl" "Fix Markup" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should add the XML declaration when
outputting XML or XHTML. Note that if the input already includes an <?xml
... ?> declaration then this option will be ignored.")

          ("add-xml-pi" "Fix Markup" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option is the same as the add-xml-decl option.")

          ("add-xml-space" "Fix Markup" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should add xml:space=\"preserve\" to elements
such as <PRE>, <STYLE> and <SCRIPT> when generating XML. This is needed if
the whitespace in such elements is to be parsed appropriately without
having access to the DTD.")

          ("alt-text" "Fix Markup" "String" ""
           "
Type: String
Default: -none-

This option specifies the default \"alt=\" text Tidy uses for <IMG>
attributes. This feature is dangerous as it suppresses further
accessibility warnings. You are responsible for making your documents
accessible to people who can not see the images!")

          ("ascii-chars" "Fix Markup" "Boolean" "yes"
           "
Type: Boolean
Default: yes
Example: y/n, yes/no, t/f, true/false, 1/0

Can be used to modify behavior of -c (--clean yes) option.
Defaults to \"yes\" when using -c. Set to \"no\" to prevent
converting &emdash;, &rdquo;, and other named character entities
```

```
to their ascii equivalents.")

          ("assume-xml-procins" "Fix Markup" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should change the parsing of processing
instructions to require ?> as the terminator rather than >. This option is
automatically set if the input is in XML.")

          ("bare" "Fix Markup" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should strip Microsoft specific HTML from
Word 2000 documents, and output spaces rather than non-breaking spaces
where they exist in the input.")

          ("break-before-br" "Fix Markup" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should output a line break before each <BR>
element.")

          ("char-encoding" "Encoding" "Encoding" "ascii"
           "
Type: Encoding
Default: ascii
Example: ascii, latin1, raw, utf8, iso2022, mac, win1252

This option specifies the character encoding Tidy uses for both the input
and output. Possible values are: ascii, latin1, raw, utf8, iso2022, mac,
win1252. For ascii, Tidy will accept Latin-1 (ISO-8859-1) character
values, but will use entities for all characters whose value > 127. For
raw, Tidy will output values above 127 without translating them into
entities. For latin1, characters above 255 will be written as
entities. For utf8, Tidy assumes that both input and output is encoded as
UTF-8. You can use iso2022 for files encoded using the ISO-2022 family of
encodings e.g. ISO-2022-JP. For mac and win1252, Tidy will accept vendor
specific character values, but will use entities for all characters whose
value > 127.")

          ("clean" "Fix Markup" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should strip out surplus presentational tags
and attributes replacing them by style rules and structural markup as
appropriate. It works well on the HTML saved by Microsoft Office products.")

          ("doctype" "Fix Markup" "DocType" "auto"
           "
Type: DocType
Default: auto
Example: auto, omit, strict, loose, transitional, user specified fpi \(string\)
```

```
This option specifies the DOCTYPE declaration generated by Tidy. If set to
\"omit\" the output won't contain a DOCTYPE declaration. If set to \"auto\"
\(the default\) Tidy will use an educated guess based upon the contents of
the document. If set to \"strict\", Tidy will set the DOCTYPE to the strict
DTD. If set to \"loose\", the DOCTYPE is set to the loose \(transitional\)
DTD. Alternatively, you can supply a string for the formal public
identifier \(FPI\). For example:

      doctype: \"-//ACME//DTD HTML 3.14159//EN\"

If you specify the FPI for an XHTML document, Tidy will set
the system identifier to the empty string. Tidy leaves the DOCTYPE for
generic XML documents unchanged.")

          ("drop-empty-paras" "Fix Markup" "Boolean" "yes"
           "
Type: Boolean
Default: yes
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should discard empty paragraphs. If set to
no, empty paragraphs are replaced by a pair of <BR> elements as HTML4
precludes empty paragraphs.")

          ("drop-font-tags" "Fix Markup" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should discard <FONT> and <CENTER> tags
rather than creating the corresponding style rules, but only if the clean
option is also set to yes.")

          ("drop-proprietary-attributes" "Fix Markup" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should strip out proprietary attributes,
such as MS data binding attributes.")

          ("enclose-block-text" "Fix Markup" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should insert a <P> element to enclose any
text it finds in any element that allows mixed content for HTML
transitional but not HTML strict.")

          ("enclose-text" "Fix Markup" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should enclose any text it finds in the body
element within a <P> element. This is useful when you want to take
existing HTML and use it with a style sheet.")

          ("error-file" "Omit" "String" "-none-"
           "
```

```
Type: String
Default: -none-

This option specifies the error file Tidy uses for errors and
warnings. Normally errors and warnings are output to \"stderr\".

This is option is ignored in Emacs.")

          ("escape-cdata" "Fix Markup" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should convert <![CDATA[]]> sections to
normal text.")

          ("fix-backslash" "Fix Markup" "Boolean" "yes"
           "
Type: Boolean
Default: yes
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should replace backslash characters \"\\\" in
URLs by forward slashes \"/\".")

          ("fix-bad-comments" "Fix Markup" "Boolean" "yes"
           "
Type: Boolean
Default: yes
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should replace unexpected hyphens with \"=\"
characters when it comes across adjacent hyphens. The default is yes. This
option is provided for users of Cold Fusion which uses the comment syntax:
<!--- --->")

          ("fix-uri" "Fix Markup" "Boolean" "yes"
           "
Type: Boolean
Default: yes
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should check attribute values that carry
URIsfor illegal characters and if such are found, escape them as HTML 4
recommends.")

          ("force-output" "Input/Output" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should produce output even if errors are
encountered. Use this option with care - if Tidy reports an error,
this means Tidy was not able to, or is not sure how to, fix the error,
so the resulting output may not reflect your intention.")

          ("gnu-emacs" "Omit" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should change the format for reporting
```

```
errors and warnings to a format that is more easily parsed by GNU
Emacs.

This option is automatically set in Emacs."  )

          ("hide-comments" "Fix Markup" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should print out comments.")

          ("hide-endtags" "Fix Markup" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should omit optional end-tags when
generating the pretty printed markup. This option is ignored if you are
outputting to XML.")

          ("indent" "Indentation" "AutoBool" "no"
           "
Type: AutoBool
Default: no
Example: auto, y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should indent block-level tags.  If set to
\"auto\", this option causes Tidy to decide whether or not to indent the
content of tags such as TITLE, H1-H6, LI, TD, TD, or P depending on whether
or not the content includes a block-level element. You are advised to avoid
setting indent to yes as this can expose layout bugs in some browsers.")

          ("indent-attributes" "Indentation" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should begin each attribute on a new line.")

          ("indent-cdata" "Indent" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should indent <![CDATA[]]> sections.")

          ("indent-spaces" "Indentation" "Integer" "2"
           "
Type: Integer
Default: 2
Example: 0, 1, 2, ...

This option specifies the number of spaces Tidy uses to indent content,
when indentation is enabled.")

          ("input-encoding" "Encoding" "Encoding" "latin1"
           "
Type: Encoding
Default: ascii
Example: ascii, latin1, raw, utf8, iso2022, mac, win1252
```

```
This option specifies the character encoding Tidy uses for the input. See
char-encoding for more info.")

          ("input-xml" "Input/Output" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should use the XML parser rather than the
error correcting HTML parser.")

          ("join-classes" "Fix Markup" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should combine class names to generate a
single new class name, if multiple class assignments are detected on an
element.")

          ("join-styles" "Fix Markup" "Boolean" "yes"
           "
Type: Boolean
Default: yes
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should combine styles to generate a single
new style, if multiple style values are detected on an element.")

          ("keep-time" "Preference" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should alter the last modified time for
files it writes back to. The default is no, which allows you to tidy files
without affecting which ones will be uploaded to a Web server when using a
tool such as 'SiteCopy'. Note that this feature may not work on some
platforms.")

          ("literal-attributes" "Preference" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should ensure that whitespace characters
within attribute values are passed through unchanged.")

          ("logical-emphasis" "Fix Markup" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should replace any occurrence of <I> by <EM>
and any occurrence of <B> by <STRONG>. In both cases, the attributes are
preserved unchanged. This option can be set independently of the clean and
drop-font-tags options.")

          ("lower-literals" "Preference" "Boolean" "yes"
          "
Type: Boolean
Default: yes
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should convert the value of an attribute
that takes a list of predefined values to lower case. This is required for
XHTML documents.")

          ("markup" "Input/Output" "Boolean" "yes"
          "
Type: Boolean
Default: yes
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should generate a pretty printed version of
the markup. Note that Tidy won't generate a pretty printed version if it
finds significant errors (see force-output).")

          ("ncr" "Preference" "Boolean" "yes"
          "
Type: Boolean
Default: yes
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should allow numeric character references.")

          ("new-blocklevel-tags" "Tags" "Tag names" ""
          "
Type: Tag names
Default: -none-
Example: tagX, tagY, ...

This option specifies new block-level tags. This option takes a space or
comma separated list of tag names. Unless you declare new tags, Tidy will
refuse to generate a tidied file if the input includes previously unknown
tags. Note you can't change the content model for elements such as
<TABLE>, <UL>, <OL> and <DL>.")

          ("new-empty-tags" "Tags" "Tag names" ""
          "
Type: Tag names
Default: -none-
Example: tagX, tagY, ...

This option specifies new empty inline tags. This option takes a space or
comma separated list of tag names. Unless you declare new tags, Tidy will
refuse to generate a tidied file if the input includes previously unknown
tags. Remember to also declare empty tags as either inline or blocklevel.")

          ("new-inline-tags" "Tags" "Tag names" ""
          "
Type: Tag names
Default: -none-
Example: tagX, tagY, ...

This option specifies new non-empty inline tags. This option takes a space
or comma separated list of tag names. Unless you declare new tags, Tidy
will refuse to generate a tidied file if the input includes previously
unknown tags.")

          ("new-pre-tags" "Tags" "Tag names" ""
          "
Type: Tag names
Default: -none-
```

```
Example: tagX, tagY, ...

This option specifies new tags that are to be processed in exactly the
same way as HTML's <PRE> element. This option takes a space or comma
separated list of tag names. Unless you declare new tags, Tidy will refuse
to generate a tidied file if the input includes previously unknown
tags. Note you can not as yet add new CDATA elements (similar to
<SCRIPT>).")

          ("numeric-entities" "Preference" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should output entities other than the
built-in HTML entities \(&amp;, &lt;, &gt; and &quot;\) in the numeric
rather than the named entity form.")

          ("output-bom" "Encoding" "AutoBool" "auto"
           "
Type: AutoBool
Default: auto
Example: auto, y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should write a Unicode Byte Order Mark
character (BOM; also known as Zero Width No-Break Space; has value of
U+FEFF) to the beginning of the output; only for UTF-8 and UTF-16 output
encodings. If set to \"auto\", this option causes Tidy to write a BOM to the
output only if a BOM was present at the beginning of the input. A BOM is
always written for XML/XHTML output using UTF-16 output encodings.")

          ("output-encoding" "Encoding" "Encoding" "ascii"
           "
Type: Encoding
Default: ascii
Example: ascii, latin1, raw, utf8, iso2022, mac, win1252

This option specifies the character encoding Tidy uses for the output. See
char-encoding for more info. May only be different from input-encoding for
Latin encodings (ascii, latin1, mac, win1252).")

          ("output-xhtml" "Input/Output" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should generate pretty printed output,
writing it as extensible HTML. This option causes Tidy to set the DOCTYPE
and default namespace as appropriate to XHTML. If a DOCTYPE or namespace
is given they will checked for consistency with the content of the
document. In the case of an inconsistency, the corrected values will
appear in the output. For XHTML, entities can be written as named or
numeric entities according to the setting of the \"numeric-entities\"
option.  The original case of tags and attributes will be preserved,
regardless of other options.")

          ("output-xml" "Input/Output" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should pretty print output, writing it as
```

```
well-formed XML. Any entities not defined in XML 1.0 will be written as
numeric entities to allow them to be parsed by a XML parser. The original
case of tags and attributes will be preserved, regardless of other
options.")

          ("quiet" "Input/Output" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should output the summary of the numbers of
errors and warnings, or the welcome or informational messages.")

          ("quote-ampersand" "Preference" "Boolean" "yes"
           "
Type: Boolean
Default: yes
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should output unadorned & characters as
&amp;.")

          ("quote-marks" "Preference"  "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should output \" characters as &quot; as is
preferred by some editing environments. The apostrophe character \' is
written out as &#39; since many web browsers don't yet support &apos;.")

          ("quote-nbsp" "Preference" "Boolean" "yes"
           "
Type: Boolean
Default: yes
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should output non-breaking space characters
as entities, rather than as the Unicode character value 160 (decimal).")

          ("raw" "Omit" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0 char-encoding

Currently not used, but this option would be the same as the
char-encoding: raw option.")

          ("repeated-attributes" "Fix Markup" ("keep-first" "keep-last") "keep-last"
           "
Type: -
Default: keep-last
Example: keep-first, keep-last

This option specifies if Tidy should keep the first or last attribute, if
an attribute is repeated, e.g. has two align attributes.")


          ("replace-color" "Fix Markup" "Boolean"
           "
Type: Boolean
Default: no
```

```
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should replace numeric values in color
attributes by HTML/XHTML color names where defined, e.g. replace
\"#ffffff\" with \"white\"."  )

          ("slide-style" "Omit" "String"
           "
Type: Name
Default: -none-
split Currently not used.")

          ("show-body-only" "Omit" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should print only the contents of the body
tag as an HTML fragment. Useful for incorporating existing whole pages as
a portion of another page.

Emacs overrides this option.")

          ("show-errors" "Input/Output" "Integer" "6"
           "
Type: Integer
Default: 6
Example: 0, 1, 2, ...

This option specifies the number Tidy uses to determine if further errors
should be shown. If set to 0, then no errors are shown.")

          ("show-warnings" "Input/Output" "Boolean" "yes"
           "
Type: Boolean
Default: yes
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should suppress warnings. This can be useful
when a few errors are hidden in a flurry of warnings.")

          ("slide-style" "Omit" "String" ""
           "
Type: Name
Default: -none-

Currently not used.")

          ("split" "Omit" "Boolean" "no"
            "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should create a sequence of slides from
the input, splitting the markup prior to each successive <H2>. The
slides are written to \"slide001.html\", \"slide002.html\" etc.

There is currently no Emacs support for this option.")

          ("tab-size" "Indentation" "Integer" "4"
           "
Type: Integer
Default: 4
```

```
Example: 0, 1, 2, ...

This option specifies the number of columns that Tidy uses between
successive tab stops. It is used to map tabs to spaces when reading the
input. Tidy never outputs tabs.")

          ("tidy-mark" "Preference" "Boolean" "yes"
           "
Type: Boolean
Default: yes
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should add a meta element to the document
head to indicate that the document has been tidied. Tidy won't add a meta
element if one is already present.")

          ("uppercase-attributes" "Preference" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should output attribute names in upper
case. The default is no, which results in lower case attribute names,
except for XML input, where the original case is preserved.")

          ("uppercase-tags" "Preference" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should output tag names in upper case. The
default is no, which results in lower case tag names, except for XML
input, where the original case is preserved.")

          ("word-2000" "Fix Markup" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should go to great pains to strip out all
the surplus stuff Microsoft Word 2000 inserts when you save Word documents
as \"Web pages\".  Doesn't handle embedded images or VML.")

          ("wrap" "Line Wrapping" "Integer" "68"
           "
Type: Integer
Default: 68
Example: 0, 1, 2, ...

This option specifies the right margin Tidy uses for line wrapping. Tidy
tries to wrap lines so that they do not exceed this length. Set wrap to
zero if you want to disable line wrapping.")

          ("wrap-asp" "Line Wrapping" "Boolean" "yes"
           "
Type: Boolean
Default: yes
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should line wrap text contained within ASP
pseudo elements, which look like: <% ... %>.")
```

```
          ("wrap-attributes" "Line Wrapping" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should line wrap attribute values, for
easier editing. This option can be set independently of
wrap-script-literals.")

          ("wrap-jste" "Line Wrapping" "Boolean" "yes"
           "
Type: Boolean
Default: yes
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should line wrap text contained within JSTE
pseudo elements, which look like: <# ... #>.")

          ("wrap-php" "Line Wrapping" "Boolean" "yes"
           "
Type: Boolean
Default: yes
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should line wrap text contained within PHP
pseudo elements, which look like: <?php ... ?>.")

          ("wrap-script-literals" "Line Wrapping" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should line wrap string literals that appear
in script attributes. Tidy wraps long script string literals by inserting
a backslash character before the line break.")

          ("wrap-sections" "Line Wrapping" "Boolean" "yes"
           "
Type: Boolean
Default: yes
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should line wrap text contained within
<![... ]> section tags.")

          ("write-back" "Omit" "Boolean" "no"
           "
Type: Boolean
Default: no
Example: y/n, yes/no, t/f, true/false, 1/0

This option specifies if Tidy should write back the tidied markup to
the same file it read from. You are advised to keep copies of
important files before tidying them, as on rare occasions the result
may not be what you expect.

This option is ignored by Emacs.")
          ))
  )

;;;;; Create a variable for each option in `tidy-options-alist'"

;; these variables are made buffer local so that different buffers can
;; use a different set of options

(let ((options-alist tidy-options-alist)
      option name symbol docstring)

  (while (setq option (car options-alist))
    (setq name      (nth 0 option)
          docstring (nth 4 option)
          symbol    (intern (concat "tidy-" name)))
    ;; create the variable on the fly
    (put symbol 'variable-documentation docstring)
    (make-variable-buffer-local symbol)
    (set symbol nil) ;;default)
    (setq options-alist (cdr options-alist))
    )
  )

;;;;; Menu Lock (adapted from printing.le)

;; quite compiler
(eval-when-compile
  (progn
         (or (boundp 'current-menubar)
             (defvar current-menubar nil))
         (or (fboundp 'tidy-menu-position)
             (defun  tidy-menu-position () ""))
         (or (fboundp 'tidy-menu-lock)
             (defun tidy-menu-lock (entry state path) ""))
         (or (fboundp 'tidy-menu-lookup)
             (defun tidy-menu-lookup (path) ""))
         ))

;; always define these
(defvar tidy-menu-position nil)
(defvar tidy-menu-state nil)

(cond
 ((tidy-xemacs-p)
  ;; XEmacs
  (defun tidy-menu-position ()
    (tidy-x-make-event
     'button-release
     (list 'button 1
           'x tidy-menu-x-position
           'y -5
           )))

  ;; XEmacs
  (defun tidy-menu-lock (entry state path)
    (when (and (not (interactive-p)) tidy-menu-lock)
      (or (and tidy-menu-position (eq state tidy-menu-state))
          (setq tidy-menu-position (tidy-menu-position)
                tidy-menu-state    state))
      (let* ((menu   (tidy-menu-lookup path))
             (result (tidy-x-get-popup-menu-response menu tidy-menu-position)))
        (and (tidy-x-misc-user-event-p result)
             (funcall (tidy-x-event-function result) (tidy-x-event-object result))))
      (setq tidy-menu-position nil)))

  ;; XEmacs
  (defun tidy-menu-lookup (path)
    (car (tidy-x-find-menu-item current-menubar (cons "Tidy" path)))))

 (t
    ;; GNU Emacs
  (defun tidy-menu-position ()
    (let ()
      (list (list tidy-menu-x-position '-5)
            (selected-frame))))         ; frame

  ;; GNU Emacs
  (defun tidy-menu-lock (entry state path)
    (when (and (not (interactive-p)) tidy-menu-lock)
      (or (and tidy-menu-position (eq state tidy-menu-state))
          (setq tidy-menu-position (tidy-menu-position )
                tidy-menu-state    state))
      (let* ((menu   (tidy-menu-lookup path))
             (result (x-popup-menu tidy-menu-position menu)))
        (and result
             (let ((command (lookup-key menu (vconcat result))))
               (if (fboundp command)
                   (funcall command)
                 (eval command)))))
      (setq tidy-menu-position nil)))

  ;; GNU Emacs
  (defun tidy-menu-lookup (dummy) (lookup-key (current-local-map) [menu-bar Tidy])))
 )

;;;;; Define classes of menu items

(defun tidy-set (var-sym value mess entry state &optional path)
  "Set the value of the symbol VAR-SYM to VALUE giving a message
derived from VALUE and MESS.  Pass on menu data to `tidy-menu-lock'."
  (set var-sym value)
  (message "%s is %s" mess value)
  (tidy-menu-lock entry state path))

(defun tidy-boolean-entry (symbol name type default menu)
  "Returns a menu entry that allows us to toggle the value of SYMBOL.
SYMBOL refers to the option called NAME which has default value
DEFAULT.  TYPE should always have the value \"Boolean\".  MENU refers
to the sub-menu that this item belongs to and POSITION its position in
that list."
  (cond ((equal default "no")
         (list (vector name
                       (list 'tidy-set (list 'quote symbol)
                             (list 'if symbol 'nil "yes")
                             name
                             (list 'quote menu)
                             '(quote toggle)
                             )
                       :style 'toggle
                       :selected (list 'if symbol 't 'nil))))

        ((equal default "yes")
         (list (vector name (list 'tidy-set (list 'quote symbol)
                                  (list 'if symbol 'nil "no")
                                  name
                                  (list 'quote menu)
                                  '(quote toggle)
                                  )
                       :style 'toggle
                       :selected (list 'if symbol 'nil 't))))))

(defun tidy-list-entry (symbol name type default menu)
"Returns a menu entry that allows us to set via a radio button the
value of SYMBOL.  SYMBOL refers to the option called NAME which has
default value DEFAULT.  TYPE should be a list of the possible
values. MENU refers to the sub-menu that this item belongs to and
```

```
POSITION its position in that list."
  (let (value element)
      (while (setq value (car type))
        (if (equal value default)
            (setq element
                  (append element
                          (list
                           (vector
                            (concat name " is \"" value "\"")
                            (list 'tidy-set (list 'quote symbol)
                                  (list 'if symbol 'nil value)
                                  name
                                  (list 'quote menu)
                                  '(quote toggle)
                                  )
                            :style 'radio
                            :selected (list 'if symbol 'nil 't)
                            ))))
          (setq element
                (append element
                        (list
                         (vector
                          (concat name " is \"" value "\"")

                          (list 'tidy-set (list 'quote symbol)
                                (list 'if symbol 'nil value)
                                name
                                (list 'quote menu)
                                '(quote toggle)
                                )

                          :style 'radio
                          :selected (list
                                     'if (list 'string-equal symbol value)
                                     't 'nil)
                         )))))
        (setq type (cdr type)))
        element))

(defun tidy-set-string (symbol name default)
  "Set the value of SYMBOL identified by name to VALUE,
unless VALUE equals DEFAULT, in which case we set it to nil."
  (interactive)
  (let* ((last-value (symbol-value symbol))
         (new-value
          (if (tidy-xemacs-p)
              (read-string (format "Set %s to: " name)
                           (or last-value default) nil) ;; no default arg
            (read-string (format "Set %s to: " name)
                         (or last-value default) nil default))))
    (set symbol (if (equal new-value default) nil new-value))))

(defun tidy-set-integer (symbol name default)
  "Set the value of SYMBOL identified by name to VALUE,
unless VALUE = DEFAULT, in which case we set it to nil."
  (interactive)
  (let* ((last-value (symbol-value symbol))
         ;; careful to interpret the string as a number
         (new-value
          (string-to-number
           (if (tidy-xemacs-p)
               (read-string (format "Set %s to: " name)
                (or last-value default) nil)
             (read-string (format "Set %s to: " name)
                (or last-value default) nil default)
              ))))
          (set symbol (if (= new-value (string-to-number default)) nil
                          (number-to-string new-value)))))


(defun tidy-string-entry (symbol name type default menu)
  "Returns a menu entry that allows us to set the value of SYMBOL.
SYMBOL refers to the option called NAME which has default value
DEFAULT.  TYPE should always be one of \"String\" \"Tags\", or
\"DocType\".  MENU and POSITION are not used in this case."

  (list (vector (concat "set " name)
                (list 'tidy-set-string
                      (list 'quote symbol)
                      name default))))

(defun tidy-integer-entry (symbol name type default menu)
"Returns a menu entry that allows us to set the value of SYMBOL.
SYMBOL refers to the option called NAME which has default value
DEFAULT.  TYPE should always have the value \"Integer\". MENU and
POSITION are not used in this case. "
  (list (vector (concat "set " name)
                (list 'tidy-set-integer
                      (list 'quote symbol)
                      name default))))

(defvar tidy-top-menu nil
  "The first part of teh menu.")
(when (or (null tidy-top-menu) tidy-debug)
  (setq tidy-top-menu
         '("Tidy"
           ["Tidy buffer" tidy-buffer
            :active (and tidy-shell-command ;; check that tidy can be executed
                         (file-executable-p (car (split-string tidy-shell-
command))))]

           ["Tidy region" (tidy-buffer 1)
            :active (and tidy-shell-command ;; check that tidy can be executed
                         (file-executable-p (car (split-string tidy-shell-command))))
            :keys "C-u \\[tidy-buffer]"]

           "----------------------------------------------"

           ["Parse config file" tidy-parse-config-file
            :active (and tidy-config-file (file-exists-p tidy-config-file))]

           ["Save settings" tidy-save-settings
            :active (and tidy-config-file (file-exists-p tidy-config-file))]

           "----------------------------------------------"

           ["Menu Lock" (tidy-set 'tidy-menu-lock
                                  (if tidy-menu-lock nil t)
                                  "Menu Lock"
                                  'top
                                  'toggle)
            :style toggle
            :selected (if tidy-menu-lock t nil)
            ]
           )))

(defvar tidy-doctype-menu nil "The second to last sub-menu.")
(when (or (null tidy-doctype-menu) tidy-debug)
  (setq  tidy-doctype-menu
         '("Set doctype" ;; ==>
           ["auto"   (tidy-set 'tidy-doctype
                               nil
                               "doctype"
                               'doctype
                               'toggle)
            :style radio
            :selected (if (null tidy-doctype)  t nil)]

           ["omit"   (tidy-set 'tidy-doctype
                               "omit"
                               "doctype"
                               'doctype
                               'toggle)
            :style radio
            :selected (if (equal tidy-doctype "omit") t nil)]

           ["strict" (tidy-set 'tidy-doctype
                               "strict"
                               "doctype"
                               'doctype
                               'toggle)
            :style radio
            :selected (if (equal tidy-doctype "strict") t nil)]

           ["loose"  (tidy-set 'tidy-doctype
                               "loose"
                               "doctype"
                               'doctype
                               'toggle)
            :style radio
            :selected (if (equal tidy-doctype "loose") t nil)]

           ["transitional"  (tidy-set 'tidy-doctype
                                      "transitional"
                                      "doctype"
                                      'doctype
                                      'toggle)
            :style radio
            :selected (if (equal tidy-doctype "transitional") t nil)]

           ["fpi" (null nil) ;; stub function
            :style radio
            :selected (if (or (null tidy-doctype)
                              (equal tidy-doctype "omit")
                              (equal tidy-doctype "strict")
                              (equal tidy-doctype "loose"))
                          nil t) ]

           ["reset fpi" (tidy-set-string 'tidy-doctype "doctype" "" "")]
           )))

(defvar tidy-char-encoding-menu nil "The last sub-menu.")
(when (or (null tidy-char-encoding-menu) tidy-debug)
  (setq tidy-char-encoding-menu
        '("Set char-encoding" ;; ==>
          ["ascii"   (tidy-set 'tidy-char-encoding
                               nil
                               "char-encoding"
                               'encoding
                               'toggle)
           :style radio
           :selected (if (null tidy-char-encoding) t nil) ;; default
           ]
          ["raw"     (tidy-set 'tidy-char-encoding
                               "raw"
                               "char-encoding"
                               'encoding
                               'toggle)
           :style radio
           :selected (if (equal tidy-char-encoding "raw") t nil)]

          ["latin1"  (tidy-set 'tidy-char-encoding
                               "latin1"
                               "char-encoding"
                               'encoding
                               'toggle)
           :style radio
           :selected (if (equal tidy-char-encoding "latin1") t nil)]

          ["utf8"    (tidy-set 'tidy-char-encoding
                               "utf8"
                               "char-encoding"
                               'encoding
                               'toggle)
           :style radio
           :selected (if (equal tidy-char-encoding "utf8") t nil)]

          ["iso2022" (tidy-set 'tidy-char-encoding
                               "iso2022"
                               "char-encoding"
                               'encoding
                               'toggle)
           :style radio
           :selected (if (equal tidy-char-encoding "iso2022") t nil)]

          ["mac" (tidy-set 'tidy-char-encoding
                           "mac"
                           "char-encoding"
                           'encoding
                           'toggle)
           :style radio
           :selected (if (equal tidy-char-encoding "mac") t nil)]

          ["win1252" (tidy-set 'tidy-char-encoding
                               "win1252"
                               "char-encoding"
                               'encoding
                               'toggle)
           :style radio
           :selected (if (equal tidy-char-encoding "win1252") t nil)]

          )))

;;;;; Create a menu item for each option that has a valid sub-menu
;; field

(defvar tidy-menu nil "Menu used by tidy.")
(when (or (null tidy-menu) tidy-debug)
  (let ((options-alist       tidy-options-alist)

        ;; sub menus are divided into two parts with list type options
        ;; coming first, followed by the rest

        markup-menu-bool     markup-menu-set
        line-wrap-menu-bool  line-wrap-menu-set
        preference-menu-bool preference-menu-set
        indent-menu-bool     indent-menu-set
        io-menu-bool         io-menu-set
```

```
         tags-menu-bool       tags-menu-set

         name sub-menu type default symbol entry entry-function option)

    (while (setq option (car options-alist))
      (setq name      (nth 0 option)
            sub-menu  (nth 1 option)
            type      (nth 2 option)
            default   (nth 3 option)
            symbol    (intern (concat "tidy-" name))
            entry     nil)

      (cond ((equal type "Boolean")
             (setq entry-function 'tidy-boolean-entry))

            ((equal type "AutoBool")
             (setq entry-function 'tidy-list-entry)
             (setq type '("auto" "yes" "no")))

            ((equal type "DocType")
             (setq entry '())) ;; handled below

            ((equal type "Tag names")
             (setq entry-function 'tidy-string-entry))

            ((equal type "String")
             (setq entry-function 'tidy-string-entry))

            ((equal type "Integer")
             (setq entry-function 'tidy-integer-entry))

            ((equal type "Encoding")
             (setq entry '()));; handled below

            ((listp type)
             (setq entry-function 'tidy-list-entry))
            (t
             (error (concat "Tidy: unhandled value type " type " for " name))))

      (cond ((equal sub-menu "Fix Markup")
             (setq entry (funcall
                          entry-function
                          symbol
                          name
                          type
                          default
                          'markup))

             (if (or (equal type "Boolean") (equal type "AutoBool") (listp type))
                 (setq markup-menu-bool (append markup-menu-bool entry))
               (setq markup-menu-set (append markup-menu-set entry))))

            ((equal sub-menu "Indentation")
             (setq entry (funcall
                          entry-function
                          symbol
                          name
                          type
                          default
                          'indent))

             (if (or (equal type "Boolean") (equal type "AutoBool") (listp type))
                 (setq indent-menu-bool (append indent-menu-bool entry))
               (setq indent-menu-set (append indent-menu-set entry))))
            ((equal sub-menu "Line Wrapping")
             (setq entry (funcall
                          entry-function
                          symbol
                          name
                          type
                          default
                          'line-wrap))

             (if (or (equal type "Boolean") (equal type "AutoBool") (listp type))
                 (setq line-wrap-menu-bool (append line-wrap-menu-bool entry))
               (setq line-wrap-menu-set (append line-wrap-menu-set entry))))

            ((equal sub-menu "Input/Output")
             (setq entry (funcall
                          entry-function
                          symbol
                          name
                          type
                          default
                          'io))

             (if (or (equal type "Boolean") (equal type "AutoBool") (listp type))
                 (setq io-menu-bool (append io-menu-bool entry))
               (setq io-menu-set (append io-menu-set entry))))

            ((equal sub-menu "Preference")
             (setq entry (funcall
                          entry-function
                          symbol
                          name
                          type
                          default
                          'preference))

             (if (or (equal type "Boolean") (equal type "AutoBool") (listp type))
                 (setq preference-menu-bool (append preference-menu-bool entry))
               (setq preference-menu-set (append preference-menu-set entry))))

            ((equal sub-menu "Tags")
             (setq entry (funcall
                          entry-function
                          symbol
                          name
                          type
                          default
                          'tags))

             (if (or (equal type "Boolean") (equal type "AutoBool"))
             (setq tags-menu-bool (append tags-menu-bool entry))
             (setq tags-menu-set (append tags-menu-set entry))))
            (t)) ;; we simple omit all other menus

      (setq options-alist (cdr options-alist)))

  (setq tidy-menu (append
                   tidy-top-menu
                   (list (append (list "Fix Markup")
                               markup-menu-bool
                               markup-menu-set))
                   (list (append  (list "Line Wrapping")
                               line-wrap-menu-bool
                               line-wrap-menu-set))
                   (list (append (list "Preference")
                               preference-menu-bool
```

```
                        preference-menu-set))
                  (list (append (list "Indentation")
                        indent-menu-bool
                        indent-menu-set))
                  (list (append(list "Input/Output")
                        io-menu-bool
                        io-menu-set))
                  (list (append (list "Tags")
                        tags-menu-bool
                        tags-menu-set))
                  (list tidy-doctype-menu)
                  (list tidy-char-encoding-menu)
                  '(["Describe options" tidy-describe-options t])))
  )
)

(defun tidy-build-menu (&optional map)
  "Set up the tidy menu in MAP. Used to set up a Tidy menu in your
favourite mode."
  (interactive) ;; for debugging
  (or map (setq map (current-local-map)))
  (tidy-parse-config-file)
  (easy-menu-remove tidy-menu)
  (easy-menu-define tidy-menu-symbol map "Menu for tidy.el" tidy-menu)
  (easy-menu-add tidy-menu map))

;;;;; Option description support

;; quiet FSF Emacs and XEmacs compilers
(eval-when-compile
  (progn (or (fboundp 'event-point) (defun event-point (dummy) ""))
         (or (fboundp 'posn-point)  (defun posn-point  (dummy) ""))
         (or (fboundp 'event-start) (defun event-start (dummy) ""))))

(defun tidy-describe-this-option (click)
  "Describe variable associated with the text at point."
  (interactive "e")

  (let* ((variable (get-text-property
                    (if (tidy-xemacs-p)
                        (event-point click)
                      (posn-point (event-start click))) 'tidy-variable))
         value
         buffer) ;; reuse the help buffer
    (when variable
      (save-selected-window
        (setq value (symbol-value variable)
              buffer (get-buffer-create "*Help*"))
        (set-buffer buffer)
        (setq buffer-read-only nil)
        (delete-region (point-min) (point-max)) ;; empty the buffer
        (insert (substring (symbol-name variable) 5) ;; clip the `tidy-' prefix
                " is set to ")
        (if value (insert value) (insert "set to the default value"))

        (insert "\n\n" (documentation-property variable 'variable-documentation))
        (setq buffer-read-only t)
        (local-set-key [(q)] 'tidy-quit-describe-options)
        (pop-to-buffer buffer)))))

(defun tidy-quit-describe-options ()
"Rid thyself of any display associated with Tidy's options."
  (interactive)
  (bury-buffer (get-buffer "*tidy-options*"))
  (delete-windows-on (get-buffer "*tidy-options*"))
  (bury-buffer (get-buffer "*Help*"))
  (delete-windows-on (get-buffer "*Help*")))

;; nicked this from cal-desk-calendar.el:-)
(defun tidy-current-line ()
  "Get the current line number (in the buffer) of point."
  (interactive)
  (save-restriction
    (widen)
    (save-excursion
      (beginning-of-line)
      (1+ (count-lines 1 (point))))))

(defun tidy-describe-options ()
  "Interactively access documentation strings for `tidy-' variables."
  (interactive)
  (let ((buffer (get-buffer "*tidy-options*")))
    (if buffer (pop-to-buffer buffer)
      ;; else build it from scratch
      (setq buffer (get-buffer-create "*tidy-options*"))
      (let* ((start 0)
             (end 0)
             name
             (count 0)
             (option-alist tidy-options-alist)
             (column2a (+ (length "drop-proprietary-attributes") 3))
             (column2b (/ (window-width) 3))
             (column2 (if (> column2a column2b) column2a column2b))
             (column3 (* 2 column2))
             (start-line 0)
             (third-length 0)
             (two-third-length 0))

        (set-buffer buffer)

        (setq buffer-read-only nil)
        (delete-region (point-min) (point-max)) ;; empty the buffer

        ;; set up local bindings
        (if (tidy-xemacs-p)
            (local-set-key [(button2)] 'tidy-describe-this-option)
          (local-set-key [(mouse-2)] 'tidy-describe-this-option))

        (local-set-key [(q)] 'tidy-quit-describe-options)

        (insert "Click [mouse-2] over option to see its description.  "
                "Type \"q\" to quit." "\n\n")

        (setq start-line (tidy-current-line))
        (setq third-length (1+ (/ (length option-alist) 3) ))
        (setq two-third-length (1- (* 2 third-length)))

        (while (setq name (car (car-safe option-alist)))
          (setq option-alist (cdr option-alist))
          (setq count (+ count 1))

          (cond
           ((< count third-length)     ;; 0 <= count < third-length
            (setq start (point))
                 (insert name)
                 (setq end (point))
                 (insert "\n"))
           ((< count two-third-length) ;; third-length <= count < two-third-length
            (if (= count third-length)
                 (goto-line start-line)
```

```
              (forward-line 1))
            (end-of-line)
            (setq start (point))
            (indent-to-column column2)
            (setq end (point))
            (put-text-property start end 'mouse-face 'default)
            (setq start (point))
            (insert name)
            (setq end (point)))
           (t                          ;; two-third-length <= count < length
            (if (= count two-third-length)
                (goto-line start-line)
              (forward-line 1))
            (end-of-line)
            (setq start (point))
            (indent-to-column column3)
            (setq end (point))
            (put-text-property start end 'mouse-face 'default)
            (setq start (point))
            (insert name)
            (setq end (point))))

          ;; make the strings funky
          (put-text-property start end 'mouse-face 'highlight)
          (put-text-property start end 'tidy-variable (intern (concat "tidy-"
name)))
          )
        (setq buffer-read-only t)
        (beginning-of-buffer)
        (pop-to-buffer buffer)
        ))))

;;;;; Configuration file support

(defun tidy-parse-config-file ()
  "If `tidy-config-file' is non-nil parse that file setting variables accordingly."
  (interactive)
  (when tidy-config-file
    (if (not (file-exists-p tidy-config-file))
        (message "Could not find config file \"%s\".  Winging it." tidy-config-file)
      (message "Parsing config file...")
      (let ((html-buffer (current-buffer))
            (config-buffer (find-file-noselect tidy-config-file t)))
        (save-excursion
          (set-buffer config-buffer)
          (goto-char (point-min)) ;; unnecessary but pedantic

          ;; delete all comments
          (while (re-search-forward "//.*\n" nil t)
            (replace-match "" nil nil))

          (goto-char (point-min))
          (while (re-search-forward "\\([a-z,-]+\\):\\s-*\\(.*\\)\\s-*" nil t)
            ;; set the variable
            ;; Thanks to Thomas Baumann for this bugfix
            (let ((variable (concat "tidy-" (match-string 1)))
                  (value (match-string 2)))
              (save-excursion
                (set-buffer html-buffer)
                (set (intern variable) value))))

          (set-buffer-modified-p nil) ;; don't save changes
          (kill-buffer config-buffer)))
      (message "Parsing config file...done")
      )))
```

```
(defun tidy-save-settings (&optional config-file)
  "Query saving the current settings to your `tidy-config-file'.
Perhaps put this on your `kill-buffer-hook'."
  (interactive)
  (or config-file (setq config-file tidy-config-file))
  (when config-file

      ;; should check for locks!
      (if (or (not (interactive-p))
              (y-or-n-p "Save settings to your tidy configuration file? "))

          (let ((buffer (find-file-noselect config-file t))
                (option-alist tidy-options-alist)
                (outer-buffer (current-buffer))
                option name symbol value)
            (save-excursion
              (set-buffer buffer)
              (delete-region (point-min) (point-max)) ;; clear the buffer

              ;; need this line so that config file is always non empty
              (insert "// HTML Tidy configuration file \n")
              (while (setq option (car option-alist))
                (setq option-alist (cdr option-alist))
                (setq name      (nth 0 option)
                      symbol    (intern (concat "tidy-" name)))
                (save-excursion ;; this is a local variable
                  (set-buffer outer-buffer)
                  (setq value (symbol-value symbol)))
                (when value ;; nil iff default
                  (insert (car option) ": " value "\n")))

              (save-buffer)
              (kill-buffer buffer)
              )))))


;;;;; Main user function

(eval-when-compile (defvar tidy-markup nil ""))

(defun tidy-set-buffer-unmodified (dummy1 dummy2 dumm3)
  "Used to prevent error buffer form being marked as modified."
  (set-buffer-modified-p nil))

(defun tidy-buffer (&optional prefix)
  "Run the HTML Tidy program on the current buffer.
If PREFIX is non-nil, or if called interactively with a prefix argument,
then Tidy is applied to the currently selected region.  Any error messages
generated by that program are sent to \"*tidy-errors*\" buffer."

  (interactive "P")

  (let* ((start (if (null prefix) (point-min) (mark)))
         (end   (if (null prefix) (point-max) (point)))
         (filename (file-name-nondirectory (buffer-file-name (current-buffer))))
         (error-file "temp-tidy-errors")
         (error-buffer (get-buffer error-file))
         (temp-buffer " *tidy-temp*")  ;; invisible
         (config-file "temp-tidy-config")
         (command (concat tidy-shell-command
                          ;; load configuration file first so that
                          ;; options are overridden by command line
                          " -config " config-file
                          " --error-file " error-file
```

```
                        " --gnu-emacs yes"
                        " --write-back no"
                        (if prefix " --show-body-only yes"
                          " --show-body-only no")))
         (errors 0)
         (warnings 0)
         (tidy-message "")
         (seg-error nil))

    (if (> start end) (setq end (mark) start (point)));; rare case swap

    (when error-buffer             ;; flush the error buffer
      (save-excursion
         (set-buffer error-buffer)
         (set-buffer-modified-p nil)
         (kill-buffer error-buffer)))

    (when (get-buffer temp-buffer) ;; flush the temp buffer
      (save-excursion
        (set-buffer temp-buffer)
        (delete-region (point-min) (point-max))))

    (tidy-save-settings config-file)

    ;; OK do the tidy
    (shell-command-on-region start end command temp-buffer nil)

    ;; Since XEmacs can't grab the std error stream we use an error file
    (setq error-buffer (find-file-noselect error-file))

    ;; avoid leaving theses guys lying around
    (if (file-exists-p error-file)  (delete-file error-file))
    (if (file-exists-p config-file) (delete-file config-file))

    ;; scan the buffer for error strings
    (save-excursion
      (set-buffer error-buffer)
      (goto-char (point-min))
      (when (re-search-forward (concat
                              "\\([0-9]+\\) warnings?, "
                              "\\([0-9]+\\) errors? were found!")
                             nil t)
        (setq warnings (string-to-number (match-string 1)))
        (setq errors (string-to-number (match-string 2)))
        (setq tidy-message (match-string 0)))

      (goto-char (point-min))
      (while (re-search-forward "stdin:" nil t)
        (replace-match (concat filename ":")))
      (setq buffer-read-only t)
      (compilation-mode)
      ;; don't save changes
      (set-buffer-modified-p nil)
      ;; Unfortunately as soon as you do `next-error' this will change
      ;; the buffer again (setting text properties), so we make it
      ;; impossible to mark this buffer as modified by setting the
      ;; following buffer local variable:
      (make-variable-buffer-local 'after-change-functions)
      (add-hook 'after-change-functions 'tidy-set-buffer-unmodified t t)
;;      (remove-hook 'after-change-functions 'tidy-set-buffer-unmodified t)
      (goto-char (point-min))
      )

    ;; Catch segmentation violations
    ;; sometimes get this when editing files from Macs
    ;; See the function at the bottom of the file

    (if (buffer-live-p temp-buffer)
        (save-excursion
          (set-buffer temp-buffer)
          (goto-char (point-min))
          (let ((case-fold-search t))
            (if (looking-at "Segmentation") ;; might work with XEmacs
                (setq seg-error t)))))

    (unless (or (> errors 0) seg-error)
      (let* ((window (get-buffer-window (current-buffer)))
             (top (window-start window)))

        (unless tidy-markup ;; default is "yes" hence inverted logic
          (delete-region start end)   ;; delete the buffer/region
          (insert-buffer temp-buffer)) ;; replace with tidied text

        ;; Try not to move the window too much when we tidy the whole buffer
        (set-window-start window top)))

    ;; only pop-up window if there's an error

    (if (and (= warnings 0)
             (= errors 0))
        (delete-windows-on error-buffer t)  ;; else delete the pop-up window

;;; Thanks to Thomas Baumann for the following fix
;;;       (pop-to-buffer error-buffer t)
;;;       (beginning-of-buffer))
      ;; display the error-buffer, but do not select it
      ;; one can use C-x ` or mouse-2 to jump to the errors
      (display-buffer error-buffer t))
;;;

    (delete-windows-on temp-buffer t) ;; sometimes pops up

    (if seg-error
        (message "Tidy: Segmentation violation!!!  Check your character encoding.")
      (message "%s" tidy-message)
      )
    ))

;;;}}} +

;;;}}}

(provide 'tidy)

;;; tidy.el ends here

;;; Local Variables:
;;; mirror-update-on-save: t
;;; mirror-file-path: "~/public_html/emacs/tidy.el"
;;; auto-recompile: t
;;; End:
```