Specialized Emacs Features

This manual describes specialized features of Emacs.

Table of Contents

Introduction

This manual contains detailed information about various features that are too specialized to be included in the printed Emacs manual. It is intended to be readable by anyone having a basic knowledge of Emacs. However, certain sections may be intended for a more specialized audience, such as Elisp authors. This should be clearly pointed out at the beginning of these sections.

Certain packages (or collections of related features) have their own manuals, separate from the main Emacs manual. This manual is intended as a complement, rather than an alternative, to reading those additional manuals. In a nutshell, it is a collection of smaller specialized features (or extra detail about standard features), too small or too obscure to justify their own manual, or inclusion in the printed Emacs manual. The chapters in this manual are, however, included (at the relevant places) in the main Emacs manual when it is formatted as an Info document.

Sections intended specifically for Elisp programmers can follow the style of the Elisp manual. Other sections should follow the style of the Emacs manual.

Editing Pictures

To edit a picture made out of text characters (for example, a picture of the division of a register into fields, as a comment in a program), use the command *M-x picture-mode* to enter Picture mode.

In Picture mode, editing is based on the *quarter-plane* model of text, according to which the text characters lie studded on an area that stretches infinitely far to the right and downward. The concept of the end of a line does not exist in this model; the most you can say is where the last nonblank character on the line is found.

Of course, Emacs really always considers text as a sequence of characters, and lines really do have ends. But Picture mode replaces the most frequently-used commands with variants that simulate the quarter-plane model of text. They do this by inserting spaces or by converting tabs to spaces.

Most of the basic editing commands of Emacs are redefined by Picture mode to do essentially the same thing but in a quarter-plane way. In addition, Picture mode defines various keys starting with the *C-c* prefix to run special picture editing commands.

One of these keys, *C-c C-c*, is particularly important. Often a picture is part of a larger file that is usually edited in some other major mode. Picture mode records the name of the previous major mode so you can use the *C-c C-c* command (`picture-mode-exit`) later to go back to that mode. *C-c C-c* also deletes spaces from the ends of lines, unless given a numeric argument.

The special commands of Picture mode all work in other modes (provided the `picture` library is loaded), but are not bound to keys except in Picture mode. The descriptions below talk of moving "one column" and so on, but all the picture mode commands handle numeric arguments as their normal equivalents do.

Turning on Picture mode runs the hook `picture-mode-hook`. Additional extensions to Picture mode can be found in `artist.el`.

Basic Editing in Picture Mode

Most keys do the same thing in Picture mode that they usually do, but do it in a quarter-plane style. For example, *C-f* is rebound to run `picture-forward-column`, a command which moves point one column to the right, inserting a space if necessary so that the actual end of the line makes no difference. *C-b* is rebound to run `picture-backward-column`, which always moves point left one column, converting a tab to multiple spaces if necessary. *C-n* and *C-p* are rebound to run `picture-move-down` and `picture-move-up`, which can either insert spaces or convert tabs as necessary to make sure that point stays in exactly the same column. *C-e* runs `picture-end-of-line`, which moves to after the last nonblank character on the line. *C-a* runs `picture-beginning-of-line`. (The choice of screen model does not affect beginnings of lines; the only extra thing this command does is update the current picture column to 0.)

Insertion of text is adapted to the quarter-plane screen model through the use of Overwrite mode (see Section "Minor Modes" in *the Emacs Manual*.) Self-inserting characters replace existing text, column by column, rather than pushing existing text to the right. `RET` runs `picture-newline`, which just moves to the beginning of the following line so that new text will replace that line.

In Picture mode, the commands that normally delete or kill text, instead erase text (replacing it with spaces). `DEL` (`picture-backward-clear-column`) replaces the preceding character with a space rather than removing it; this moves point backwards. *C-d* (`picture-clear-column`) replaces the next character or characters with spaces, but does not move point. (If you want to clear characters to spaces and move forward over them, use `SPC`.) *C-k* (`picture-clear-line`) really kills the contents of lines, but does not delete the newlines from the buffer.

To do actual insertion, you must use special commands. *C-o* (`picture-open-line`) creates a blank line after the current line; it never splits a line. *C-M-o* (`split-line`) makes sense in Picture mode, so it is not changed. *C-j* (`picture-duplicate-line`) inserts another line with the same contents below the current line.

To do actual deletion in Picture mode, use *C-w*, *C-c C-d* (which is defined as `delete-char`, as *C-d* is in other modes), or one of the picture rectangle commands (see [Rectangles in Picture], page 4).

Controlling Motion after Insert

Since self-inserting characters in Picture mode overwrite and move point, there is no essential restriction on how point should be moved. Normally point moves right, but you can specify any of the eight orthogonal or diagonal directions for motion after a self-inserting character. This is useful for drawing lines in the buffer.

C-c `<`
C-c `LEFT` Move left after insertion (`picture-movement-left`).

C-c `>`
C-c `RIGHT` Move right after insertion (`picture-movement-right`).

C-c `^`
C-c `UP` Move up after insertion (`picture-movement-up`).

C-c `.`
C-c `DOWN` Move down after insertion (`picture-movement-down`).

C-c `` ` ``
C-c `Home` Move up and left ("northwest") after insertion (`picture-movement-nw`).

C-c `'`
C-c `PageUp`
C-c `prior` Move up and right ("northeast") after insertion (`picture-movement-ne`).

C-c `/`
C-c `End` Move down and left ("southwest") after insertion (`picture-movement-sw`).

C-c `\`
C-c `PageDown`
C-c `next` Move down and right ("southeast") after insertion (`picture-movement-se`).

Two motion commands move based on the current Picture insertion direction. The command *C-c C-f* (`picture-motion`) moves in the same direction as motion after insertion currently does, while *C-c C-b* (`picture-motion-reverse`) moves in the opposite direction.

Picture Mode Tabs

Two kinds of tab-like action are provided in Picture mode. Use *M*-TAB (`picture-tab-search`) for context-based tabbing. With no argument, it moves to a point underneath the next "interesting" character that follows whitespace in the previous nonblank line. "Next" here means "appearing at a horizontal position greater than the one point starts out at". With prefix argument, as in *C-u M*-TAB, this command moves to the next such interesting character in the current line. *M*-TAB does not change the text; it only moves point. "Interesting" characters are defined by the variable `picture-tab-chars`, which should define a set of characters. The syntax for this variable is like the syntax used inside of '`[...]`' in a regular expression—but without the '`[`' and the '`]`'. Its default value is `"!-~"`.

TAB itself runs `picture-tab`, which operates based on the current tab stop settings; it is the Picture mode equivalent of `tab-to-tab-stop`. Normally it just moves point, but with a numeric argument it clears the text that it moves over.

The context-based and tab-stop-based forms of tabbing are brought together by the command *C-c* TAB (`picture-set-tab-stops`). This command sets the tab stops to the positions which *M*-TAB would consider significant in the current line. The use of this command, together with TAB, can get the effect of context-based tabbing. But *M*-TAB is more convenient in the cases where it is sufficient.

It may be convenient to prevent use of actual tab characters in pictures. For example, this prevents *C-x* TAB from messing up the picture. You can do this by setting the variable `indent-tabs-mode` to `nil`.

Picture Mode Rectangle Commands

Picture mode defines commands for working on rectangular pieces of the text in ways that fit with the quarter-plane model. The standard rectangle commands may also be useful. See Section "Rectangles" in *the Emacs Manual*.

C-c C-k Clear out the region-rectangle with spaces (`picture-clear-rectangle`). With a prefix argument, delete the text.

C-c C-w r Similar, but save rectangle contents in register *r* first (`picture-clear-rectangle-to-register`). See Section "Registers" in *the Emacs Manual*.

C-c C-y Copy last killed rectangle into the buffer by overwriting, with upper left corner at point (`picture-yank-rectangle`). With argument, insert instead.

C-c C-x r Similar, but use the rectangle in register *r* (`picture-yank-rectangle-from-register`).

The picture rectangle commands *C-c C-k* (`picture-clear-rectangle`) and *C-c C-w* (`picture-clear-rectangle-to-register`) differ from the standard rectangle commands in that they normally clear the rectangle instead of deleting it; this is analogous with the way *C-d* is changed in Picture mode.

However, deletion of rectangles can be useful in Picture mode, so these commands delete the rectangle if given a numeric argument. *C-c C-k* either with or without a numeric argument saves the rectangle for *C-c C-y*.

The Picture mode commands for yanking rectangles differ from the standard ones in that they overwrite instead of inserting. This is the same way that Picture mode insertion of other text differs from other modes. `C-c C-y` (`picture-yank-rectangle`) inserts (by overwriting) the rectangle that was most recently killed, while `C-c C-x` (`picture-yank-rectangle-from-register`) does likewise for the rectangle found in a specified register.

Auto Reverting Non-File Buffers

Global Auto Revert Mode normally only reverts file buffers. There are two ways to auto-revert certain non-file buffers: by enabling Auto Revert Mode in those buffers (using `M-x auto-revert-mode`); and by setting `global-auto-revert-non-file-buffers` to a non-`nil` value. The latter enables Auto Reverting for all types of buffers for which it is implemented (listed in the menu below).

Like file buffers, non-file buffers should normally not revert while you are working on them, or while they contain information that might get lost after reverting. Therefore, they do not revert if they are modified. This can get tricky, because deciding when a non-file buffer should be marked modified is usually more difficult than for file buffers.

Another tricky detail is that, for efficiency reasons, Auto Revert often does not try to detect all possible changes in the buffer, only changes that are major or easy to detect. Hence, enabling auto-reverting for a non-file buffer does not always guarantee that all information in the buffer is up-to-date, and does not necessarily make manual reverts useless.

At the other extreme, certain buffers automatically revert every `auto-revert-interval` seconds. (This currently only applies to the Buffer Menu.) In this case, Auto Revert does not print any messages while reverting, even when `auto-revert-verbose` is non-`nil`.

Some non-file buffers can be updated reliably by file notification on their default directory; Dired buffers is an example. The major mode can indicate this by setting `buffer-auto-revert-by-notification` to a non-`nil` value in that buffer, allowing Auto Revert to avoid periodic polling. Such notification does not include changes to files in that directory, only to the directory itself.

The details depend on the particular types of buffers and are explained in the corresponding sections.

Auto Reverting the Buffer Menu

If auto-reverting of non-file buffers is enabled, the Buffer Menu (see Section “Several Buffers” in *the Emacs Manual*) automatically reverts every `auto-revert-interval` seconds, whether there is a need for it or not. (It would probably take longer to check whether there is a need than to actually revert.)

If the Buffer Menu inappropriately gets marked modified, just revert it manually using `g` and auto-reverting will resume. However, if you marked certain buffers to get deleted or to be displayed, you have to be careful, because reverting erases all marks. The fact that adding marks sets the buffer’s modified flag prevents Auto Revert from automatically erasing the marks.

Auto Reverting Dired buffers

Dired buffers only auto-revert when the file list of the buffer’s main directory changes (e.g., when a new file is added or deleted). They do not auto-revert when information about a particular file changes (e.g., when the size changes) or when inserted subdirectories change. To be sure that *all* listed information is up to date, you have to manually revert using `g`, *even* if auto-reverting is enabled in the Dired buffer. Sometimes, you might get the impression that modifying or saving files listed in the main directory actually does cause auto-reverting. This is because making changes to a file, or saving it, very often causes changes in the directory itself; for instance, through backup files or auto-save files. However, this is not guaranteed.

If the Dired buffer is marked modified and there are no changes you want to protect, then most of the time you can make auto-reverting resume by manually reverting the buffer using `g`. There is one exception. If you flag or mark files, you can safely revert the buffer. This will not erase the flags or marks (unless the marked file has been deleted, of course). However, the buffer will stay modified, even after reverting, and auto-reverting will not resume. This is because, if you flag or mark files, you may be working on the buffer and you might not want the buffer to change without warning. If you want auto-reverting to resume in the presence of marks and flags, mark the buffer non-modified using `M-~`. However, adding, deleting or changing marks or flags will mark it modified again.

Remote Dired buffers are currently not auto-reverted. Neither are Dired buffers for which you used shell wildcards or file arguments to list only some of the files. `*Find*` and `*Locate*` buffers do not auto-revert either.

Note that auto-reverting Dired buffers may not work satisfactorily on some systems.

Subdirectory Switches in Dired

You can insert subdirectories with specified `ls` switches in Dired buffers using *`C-u i`*. You can change the `ls` switches of an already inserted subdirectory at point using *`C-u l`*.

Dired preserves the switches if you revert the buffer. Deleting a subdirectory forgets about its switches.

Using `dired-undo` (see Section "Marks vs Flags" in *the Emacs Manual*) to reinsert or delete subdirectories that were inserted with explicit switches can bypass Dired's machinery for remembering (or forgetting) switches. Deleting a subdirectory using `dired-undo` does not forget its switches. When later reinserted using *`i`*, it will be reinserted using its old switches. Using `dired-undo` to reinsert a subdirectory that was deleted using the regular Dired commands (not `dired-undo`) will originally insert it with its old switches. Reverting the buffer, however, will relist it using the buffer's default switches. If any of this yields problems, you can easily correct the situation using *`C-u i`* or *`C-u l`*.

Dired does not remember the `R` switch. Inserting a subdirectory with switches that include the `R` switch is equivalent to inserting each of its subdirectories using all remaining switches. For instance, updating or killing a subdirectory that was inserted with the `R` switch will not update or kill its subdirectories.

The buffer's default switches do not affect subdirectories that were inserted using explicitly specified switches. In particular, commands such as *`s`* that change the buffer's switches do not affect such subdirectories. (They do, however, affect subdirectories without explicitly assigned switches.)

You can make Dired forget about all subdirectory switches and relist all subdirectories with the buffer's default switches using *`M-x dired-reset-subdir-switches`*. This also reverts the Dired buffer.

More advanced features of the Calendar and Diary

This section describes some of the more advanced/specialized features of the calendar and diary. It starts with some of the many ways in which you can customize the calendar and diary to suit your personal tastes.

Customizing the Calendar

The calendar display unfortunately cannot be changed from three months, but you can customize the whitespace used by setting the variables: `calendar-left-margin`, `calendar-day-header-width`, `calendar-day-digit-width`, `calendar-column-width`, and `calendar-intermonth-spacing`. To display text *between* the months, for example week numbers, customize the variables `calendar-intermonth-header` and `calendar-intermonth-text` as described in their documentation.

The variable `calendar-month-header` controls the text that appears above each month in the calendar. By default, it shows the month and year. The variable `calendar-day-header-array` controls the text that appears above each day's column in every month. By default, it shows the first two letters of each day's name.

The variable `calendar-holiday-marker` specifies how to mark a date that is a holiday. Its value may be a single-character string to insert next to the date, or a face name to use for displaying the date. Likewise, the variable `diary-entry-marker` specifies how to mark a date that has diary entries. The function `calendar-mark-today` uses `calendar-today-marker` to mark today's date. By default, the calendar uses faces named `holiday`, `diary`, and `calendar-today` for these purposes.

Starting the calendar runs the normal hook `calendar-initial-window-hook`. Recomputation of the calendar display does not run this hook. But if you leave the calendar with the *q* command and reenter it, the hook runs again.

The variable `calendar-today-visible-hook` is a normal hook run after the calendar buffer has been prepared with the calendar, when the current date is visible in the window. One use of this hook is to mark today's date; to do that use either of the functions `calendar-mark-today` or `calendar-star-date`:

```
(add-hook 'calendar-today-visible-hook 'calendar-mark-today)
```

A similar normal hook, `calendar-today-invisible-hook` is run if the current date is *not* visible in the window.

Each of the calendar cursor motion commands runs the hook `calendar-move-hook` after it moves the cursor.

Customizing the Holidays

There are several variables listing the default holidays that Emacs knows about. These are: `holiday-general-holidays`, `holiday-local-holidays`, `holiday-solar-holidays`, `holiday-bahai-holidays`, `holiday-christian-holidays`, `holiday-hebrew-holidays`, `holiday-islamic-holidays`, `holiday-oriental-holidays`, and `holiday-other-holidays`. The names should be self-explanatory; e.g., `holiday-solar-holidays` lists sun- and moon-related holidays.

You can customize these lists of holidays to your own needs, deleting or adding holidays as described below. Set any of them to `nil` to not show the associated holidays.

The general holidays are, by default, holidays common throughout the United States. In contrast, `holiday-local-holidays` and `holiday-other-holidays` are both empty by default. These are intended for system-wide settings and your individual use, respectively.

By default, Emacs does not include all the holidays of the religions that it knows, only those commonly found in secular calendars. For a more extensive collection of religious holidays, you can set any (or all) of the variables `calendar-bahai-all-holidays-flag`, `calendar-christian-all-holidays-flag`, `calendar-hebrew-all-holidays-flag`, or `calendar-islamic-all-holidays-flag` to `t`.

Each of the holiday variables is a list of *holiday forms*, each form describing a holiday (or sometimes a list of holidays). Here is a table of the possible kinds of holiday form. Day numbers and month numbers count starting from 1, but *dayname* numbers count Sunday as 0. The argument *string* is always the description of the holiday, as a string.

`(holiday-fixed` *`month day string`*`)`
: A fixed date on the Gregorian calendar.

`(holiday-float` *`month dayname k string`*
: &optional *day*) The *k*th *dayname* (*dayname*=0 for Sunday, and so on) after or before Gregorian date *month*, *day*. Negative *k* means count back from the end of the month. Optional *day* defaults to 1 if *k* is positive, and the last day of *month* otherwise.

`(holiday-chinese` *`month day string`*`)`
: A fixed date on the Chinese calendar.

`(holiday-hebrew` *`month day string`*`)`
: A fixed date on the Hebrew calendar.

`(holiday-islamic` *`month day string`*`)`
: A fixed date on the Islamic calendar.

`(holiday-julian` *`month day string`*`)`
: A fixed date on the Julian calendar.

`(holiday-sexp` *`sexp string`*`)`
: A date calculated by the Lisp expression *sexp*. The expression should use the variable `year` to compute and return the date of a holiday in the form of a list `(`*`month day year`*`)`, or `nil` if the holiday doesn't happen this year.

`(if` *`condition holiday-form`*`)`
: A holiday that happens only if *condition* is true.

`(`*`function`* [*`args`*]`)`
: A list of dates calculated by the function *function*, called with arguments *args*.

For example, suppose you want to add Bastille Day, celebrated in France on July 14 (i.e., the fourteenth day of the seventh month). You can do this as follows:

```
(setq holiday-other-holidays '((holiday-fixed 7 14 "Bastille Day")))
```

Many holidays occur on a specific day of the week, at a specific time of month. Here is a holiday form describing Hurricane Supplication Day, celebrated in the Virgin Islands on the fourth Monday in July:

```
(holiday-float 7 1 4 "Hurricane Supplication Day")
```

Here the 7 specifies July, the 1 specifies Monday (Sunday is 0, Tuesday is 2, and so on), and the 4 specifies the fourth occurrence in the month (1 specifies the first occurrence, 2 the second occurrence, −1 the last occurrence, −2 the second-to-last occurrence, and so on).

You can specify holidays that occur on fixed days of the Bahá'í, Chinese, Hebrew, Islamic, and Julian calendars too. For example,

```
(setq holiday-other-holidays
      '((holiday-hebrew 10 2 "Last day of Hanukkah")
        (holiday-islamic 3 12 "Mohammed's Birthday")
        (holiday-julian 4 2 "Jefferson's Birthday")))
```

adds the last day of Hanukkah (since the Hebrew months are numbered with 1 starting from Nisan), the Islamic feast celebrating Mohammed's birthday (since the Islamic months are numbered from 1 starting with Muharram), and Thomas Jefferson's birthday, which is 2 April 1743 on the Julian calendar.

To include a holiday conditionally, use either Emacs Lisp's `if` or the `holiday-sexp` form. For example, American presidential elections occur on the first Tuesday after the first Monday in November of years divisible by 4:

```
(holiday-sexp '(if (zerop (% year 4))
                   (calendar-gregorian-from-absolute
                    (1+ (calendar-dayname-on-or-before
                          1 (+ 6 (calendar-absolute-from-gregorian
                                  (list 11 1 year)))))))
              "US Presidential Election")
```

or

```
(if (zerop (% displayed-year 4))
    (holiday-fixed 11
           (calendar-extract-day
             (calendar-gregorian-from-absolute
               (1+ (calendar-dayname-on-or-before
                     1 (+ 6 (calendar-absolute-from-gregorian
                              (list 11 1 displayed-year)))))))
           "US Presidential Election"))
```

Some holidays just don't fit into any of these forms because special calculations are involved in their determination. In such cases you must write a Lisp function to do the calculation. To include eclipses, for example, add `(eclipses)` to `holiday-other-holidays` and write an Emacs Lisp function `eclipses` that returns a (possibly empty) list of the relevant Gregorian dates among the range visible in the calendar window, with descriptive strings, like this:

```
(((6 4 2012) "Lunar Eclipse") ((11 13 2012) "Solar Eclipse") ... )
```

Converting from the Mayan Calendar

Here are the commands to select dates based on the Mayan calendar:

g m l Move to a date specified by the long count calendar (`calendar-mayan-goto-long-count-date`).

g m n t Move to the next occurrence of a place in the tzolkin calendar (`calendar-mayan-next-tzolkin-date`).

g m p t Move to the previous occurrence of a place in the tzolkin calendar (`calendar-mayan-previous-tzolkin-date`).

g m n h Move to the next occurrence of a place in the haab calendar (`calendar-mayan-next-haab-date`).

g m p h Move to the previous occurrence of a place in the haab calendar (`calendar-mayan-previous-haab-date`).

g m n c Move to the next occurrence of a place in the calendar round (`calendar-mayan-next-calendar-round-date`).

g m p c Move to the previous occurrence of a place in the calendar round (`calendar-mayan-previous-calendar-round-date`).

To understand these commands, you need to understand the Mayan calendars. The *long count* is a counting of days with these units:

1 kin = 1 day   1 uinal = 20 kin   1 tun = 18 uinal
1 katun = 20 tun   1 baktun = 20 katun

Thus, the long count date 12.16.11.16.6 means 12 baktun, 16 katun, 11 tun, 16 uinal, and 6 kin. The Emacs calendar can handle Mayan long count dates as early as 7.17.18.13.3, but no earlier. When you use the *g m l* command, type the Mayan long count date with the baktun, katun, tun, uinal, and kin separated by periods.

The Mayan tzolkin calendar is a cycle of 260 days formed by a pair of independent cycles of 13 and 20 days. Since this cycle repeats endlessly, Emacs provides commands to move backward and forward to the previous or next point in the cycle. Type *g m p t* to go to the previous tzolkin date; Emacs asks you for a tzolkin date and moves point to the previous occurrence of that date. Similarly, type *g m n t* to go to the next occurrence of a tzolkin date.

The Mayan haab calendar is a cycle of 365 days arranged as 18 months of 20 days each, followed by a 5-day monthless period. Like the tzolkin cycle, this cycle repeats endlessly, and there are commands to move backward and forward to the previous or next point in the cycle. Type *g m p h* to go to the previous haab date; Emacs asks you for a haab date and moves point to the previous occurrence of that date. Similarly, type *g m n h* to go to the next occurrence of a haab date.

The Maya also used the combination of the tzolkin date and the haab date. This combination is a cycle of about 52 years called a *calendar round*. If you type *g m p c*, Emacs asks you for both a haab and a tzolkin date and then moves point to the previous occurrence of that combination. Use *g m n c* to move point to the next occurrence of a combination. These commands signal an error if the haab/tzolkin date combination you have typed is impossible.

Emacs uses strict completion (see Section “Completion Exit” in *the Emacs Manual*) whenever it asks you to type a Mayan name, so you don’t have to worry about spelling.

Date Display Format

You can customize the way dates are displayed in the diary, mode lines, and messages by setting `calendar-date-display-form`. This variable holds a list of expressions that can involve the variables `month`, `day`, and `year`, which are all numbers in string form, and `monthname` and `dayname`, which are both alphabetic strings. In the American style, the default value of this list is as follows:

```
((if dayname (concat dayname ", ")) monthname " " day ", " year)
```

while in the European style this value is the default:

```
((if dayname (concat dayname ", ")) day " " monthname " " year)
```

The default ISO date representation is:

```
((format "%s-%.2d-%.2d" year (string-to-number month)
         (string-to-number day)))
```

Another typical American format is:

```
(month "/" day "/" (substring year -2))
```

Time Display Format

The calendar and diary by default display times of day in the conventional American style with the hours from 1 through 12, minutes, and either '`am`' or '`pm`'. If you prefer the European style, also known in the US as military, in which the hours go from 00 to 23, you can alter the variable `calendar-time-display-form`. This variable is a list of expressions that can involve the variables `12-hours`, `24-hours`, and `minutes`, which are all numbers in string form, and `am-pm` and `time-zone`, which are both alphabetic strings. The default value is:

```
(12-hours ":" minutes am-pm
          (if time-zone " (") time-zone (if time-zone ")"))
```

Here is a value that provides European style times:

```
(24-hours ":" minutes
          (if time-zone " (") time-zone (if time-zone ")"))
```

Note that few calendar functions return a time of day (at present, only solar functions).

Customizing the Diary

Ordinarily, the diary window indicates any holidays that fall on the date of the diary entries, either in the mode line or the buffer itself. The process of checking for holidays can be slow, depending on the defined holidays. In that case, setting `diary-show-holidays-flag` to `nil` will speed up the diary display.

The variable `diary-number-of-entries` controls the number of days of diary entries to be displayed at one time. It affects the initial display when `calendar-view-diary-initially-flag` is `t`, as well as the command *`M-x diary`*. For example, a value of 1 (the default) displays only the current day's diary entries, whereas a value of 2 will also show the next day's entries. The value can also be a vector of seven integers: for example, if the value is `[0 2 2 2 2 4 1]` then no diary entries appear on Sunday, the current date's and the next day's diary entries appear Monday through Thursday, Friday through Monday's entries appear on Friday, while on Saturday only that day's entries appear.

You can customize the form of dates in your diary file by setting the variable `diary-date-forms`. This variable is a list of patterns for recognizing a date. Each date pattern is a list whose elements may be regular expressions (see Section "Regular Expressions" in *the Emacs Lisp Reference Manual*) or the symbols `month`, `day`, `year`, `monthname`, and `dayname`. All these elements serve as patterns that match certain kinds of text in the diary file. In order for the date pattern as a whole to match, all of its elements must match consecutively.

A regular expression in a date pattern matches in its usual fashion, using the standard syntax table altered so that '*' is a word constituent.

The symbols `month`, `day`, `year`, `monthname`, and `dayname` match the month number, day number, year number, month name, and day name of the date being considered. The symbols that match numbers allow leading zeros; those that match names allow capitalization and abbreviation (as specified by `calendar-month-abbrev-array` and `calendar-day-abbrev-array`). All the symbols can match '*'; since '*' in a diary entry means "any day", "any month", and so on, it should match regardless of the date being considered.

The default value of `diary-date-forms` in the American style is provided by `diary-american-date-forms`:

```
((month "/" day "[^/0-9]")
 (month "/" day "/" year "[^0-9]")
 (monthname " *" day "[^,0-9]")
 (monthname " *" day ", *" year "[^0-9]")
 (dayname "\\W"))
```

The variables `diary-european-date-forms` and `diary-iso-date-forms` provide other default styles.

The date patterns in the list must be *mutually exclusive* and must not match any portion of the diary entry itself, just the date and one character of whitespace. If, to be mutually exclusive, the pattern must match a portion of the diary entry text—beyond the whitespace that ends the date—then the first element of the date pattern *must* be `backup`. This causes the date recognizer to back up to the beginning of the current word of the diary entry, after finishing the match. Even if you use `backup`, the date pattern must absolutely not match more than a portion of the first word of the diary entry. For example, the default value of `diary-european-date-forms` is:

```
((day "/" month "[^/0-9]")
 (day "/" month "/" year "[^0-9]")
 (backup day " *" monthname "\\W+\\<\\([^*0-9]\\|\\([0-9]+[:aApP]\\)\\)")
 (day " *" monthname " *" year "[^0-9]")
 (dayname "\\W"))
```

Notice the use of `backup` in the third pattern, because it needs to match part of a word beyond the date itself to distinguish it from the fourth pattern.

Diary Entries Using non-Gregorian Calendars

As well as entries based on the standard Gregorian calendar, your diary can have entries based on Bahá'í, Chinese, Hebrew, or Islamic dates. Recognition of such entries can be time-consuming, however, and since most people don't use them, you must explicitly enable their use. If you want the diary to recognize Hebrew-date diary entries, for example, you must do this:

```
(add-hook 'diary-nongregorian-listing-hook 'diary-hebrew-list-entries)
(add-hook 'diary-nongregorian-marking-hook 'diary-hebrew-mark-entries)
```

Similarly, for Islamic, Bahá'í and Chinese entries, add `diary-islamic-list-entries` and `diary-islamic-mark-entries`, `diary-bahai-list-entries` and `diary-bahai-mark-entries`, or `diary-chinese-list-entries` and `diary-chinese-mark-entries`.

These diary entries have the same formats as Gregorian-date diary entries; except that `diary-bahai-entry-symbol` (default '`B`') must precede a Bahá'í date, `diary-chinese-entry-symbol` (default '`C`') a Chinese date, `diary-hebrew-entry-symbol` (default '`H`') a Hebrew date, and `diary-islamic-entry-symbol` (default '`I`') an Islamic date. Moreover, non-Gregorian month names may not be abbreviated (because the first three letters are often not unique). (Note also that you must use "Adar I" if you want Adar of a common Hebrew year.) For example, a diary entry for the Hebrew date Heshvan 25 could look like this:

```
HHeshvan 25 Happy Hebrew birthday!
```

and would appear in the diary for any date that corresponds to Heshvan 25 on the Hebrew calendar. And here is an Islamic-date diary entry that matches Dhu al-Qada 25:

```
IDhu al-Qada 25 Happy Islamic birthday!
```

As with Gregorian-date diary entries, non-Gregorian entries are nonmarking if preceded by `diary-nonmarking-symbol` (default '`&`').

Here is a table of commands used in the calendar to create diary entries that match the selected date and other dates that are similar in the Bahá'í, Chinese, Hebrew, or Islamic calendars:

i h d `diary-hebrew-insert-entry`

i h m `diary-hebrew-insert-monthly-entry`

i h y `diary-hebrew-insert-yearly-entry`

i i d `diary-islamic-insert-entry`

i i m `diary-islamic-insert-monthly-entry`

i i y `diary-islamic-insert-yearly-entry`

i B d `diary-bahai-insert-entry`

i B m `diary-bahai-insert-monthly-entry`

i B y `diary-bahai-insert-yearly-entry`

i C d `diary-chinese-insert-entry`

i C m `diary-chinese-insert-monthly-entry`

i C y `diary-chinese-insert-yearly-entry`

i C `a` `diary-chinese-insert-anniversary-entry`

These commands work much like the corresponding commands for ordinary diary entries: they apply to the date that point is on in the calendar window, and what they do is insert just the date portion of a diary entry at the end of your diary file. You must then insert the rest of the diary entry. The basic commands add an entry for the specific non-Gregorian date, the '`monthly`' commands for the given non-Gregorian day-within-month in every month, and the '`yearly`' commands for the given non-Gregorian day and month in every year.

Diary Display

Diary display works by preparing the list of diary entries and then running the function specified by the variable `diary-display-function`. The default value `diary-fancy-display` displays diary entries and holidays by copying them into a special buffer that exists only for the sake of display. Copying diary entries to a separate buffer provides an opportunity to change the displayed text to make it prettier—for example, to sort the entries by the dates they apply to.

Ordinarily, the fancy diary buffer does not show days for which there are no diary entries, even if that day is a holiday. If you want such days to be shown in the fancy diary buffer, set the variable `diary-list-include-blanks` to `t`.

The fancy diary buffer enables View mode (see Section "View Mode" in *the Emacs Manual*).

The alternative display method `diary-simple-display` shows the actual diary buffer, and uses invisible text to hide entries that don't apply. Holidays are shown in the mode line. The advantage of this method is that you can edit the buffer and save your changes directly to the diary file. This method is not as flexible as the fancy method, however. For example, it cannot sort entries. Another disadvantage is that invisible text can be confusing. For example, if you copy a region of text in order to paste it elsewhere, invisible text may be included. Similarly, since the diary buffer as you see it is an illusion, simply printing the buffer may not print what you see on your screen.

For this reason, there is a special command to print hard copy of the diary buffer *as it appears*; this command is *`M-x diary-print-entries`*. It works with either display method, although with the fancy display you can also print the buffer like any other. To print a hard copy of a day-by-day diary for a week, position point on the first day of the week, type *`7 d`*, and then do *`M-x diary-print-entries`*. As usual, the inclusion of the holidays slows down the display slightly; you can speed things up by setting the variable `diary-show-holidays-flag` to `nil`.

This command prepares a temporary buffer that contains only the diary entries currently visible in the diary buffer. Unlike with the simple display, the other irrelevant entries are really absent, not just hidden. After preparing the buffer, it runs the hook `diary-print-entries-hook`. The default value of this hook sends the data directly to the printer with the command `lpr-buffer` (see Section "Printing" in *the Emacs Manual*). If you want to use a different command to do the printing, just change the value of this hook. Other uses might include, for example, rearranging the lines into order by day and time.

You can edit the diary entries as they appear in the simple diary window, but it is important to remember that the buffer displayed contains the *entire* diary file, with portions of it concealed from view. This means, for instance, that the *`C-f`* (`forward-char`) command can put point at what appears to be the end of the line, but what is in reality the middle of some concealed line.

Be careful when editing the diary entries in the simple display! Inserting additional lines or adding/deleting characters in the middle of a visible line cannot cause problems, but editing at the end of a line may not do what you expect. Deleting a line may delete other invisible entries that follow it. Before editing the simple diary buffer, it is best to display the entire file with *s* (`diary-show-all-entries`).

Fancy Diary Display

The following features only work with the fancy diary display.

You can use the normal hook `diary-list-entries-hook` to sort each day's diary entries by their time of day. Here's how:

```
(add-hook 'diary-list-entries-hook 'diary-sort-entries t)
```

For each day, this sorts diary entries that begin with a recognizable time of day according to their times. Diary entries without times come first within each day. Note how the sort command is placed at the end of the hook list, in case earlier members of the list change the order of the diary entries, or add items.

You can write '`comments`' in diary entries, by setting the variables `diary-comment-start` and `diary-comment-end` to strings that delimit comments. The fancy display does not print comments. You might want to put meta-data for the use of other packages (e.g., the appointment package, see Section "Appointments" in *the Emacs Manual*) inside comments.

Your main diary file can include other files. This permits a group of people to share a diary file for events that apply to all of them. Lines in the diary file starting with `diary-include-string`:

```
#include "filename"
```

include the diary entries from the file *filename* in the fancy diary buffer. The include mechanism is recursive, so that included files can include other files, and so on (you must be careful not to have a cycle of inclusions, of course). Here is how to enable the include facility:

```
(add-hook 'diary-list-entries-hook 'diary-include-other-diary-files)
(add-hook 'diary-mark-entries-hook 'diary-mark-included-diary-files)
```

The include mechanism works only with the fancy diary display, because simple diary display shows the entries directly from your diary file.

Sexp Entries and the Fancy Diary Display

Sexp diary entries allow you to do more than just have complicated conditions under which a diary entry applies. Sexp entries should be preceded by `diary-sexp-entry-symbol` (default '`%%`') in the diary file. With the fancy diary display, sexp entries can generate the text of the entry depending on the date itself.

For example, an anniversary diary entry can insert the number of years since the anniversary date into the text of the diary entry. Thus the '`%d`' in this diary entry:

```
%%(diary-anniversary 10 31 1948) Arthur's birthday (%d years old)
```

gets replaced by the age, so on October 31, 1990 the entry appears in the fancy diary buffer like this:

```
Arthur's birthday (42 years old)
```

If the diary file instead contains this entry:

```
%%(diary-anniversary 10 31 1948) Arthur's %d%s birthday
```

the entry in the fancy diary buffer for October 31, 1990 appears like this:

```
Arthur's 42nd birthday
```

Similarly, cyclic diary entries can interpolate the number of repetitions that have occurred:

```
%%(diary-cyclic 50 1 1 2012) Renew medication (%d%s time)
```

looks like this:

```
Renew medication (5th time)
```

in the fancy diary display on September 7, 2012.

There is an early-reminder diary sexp that includes its entry in the diary not only on the date of occurrence, but also on earlier dates. For example, if you want a reminder a week before your anniversary, you can use

```
%%(diary-remind '(diary-anniversary 12 22 1968) 7) Ed's anniversary
```

and the fancy diary will show '`Ed's anniversary`' both on December 15 and on December 22.

The function `diary-date` applies to dates described by a month, day, year combination, each of which can be an integer, a list of integers, or `t` (meaning all values). For example,

```
%%(diary-date '(10 11 12) 22 t) Rake leaves
```

causes the fancy diary to show

```
Rake leaves
```

on October 22, November 22, and December 22 of every year.

The function `diary-float` allows you to describe diary entries that apply to dates like the third Friday of November, or the last Tuesday in April. The parameters are the *month*, *dayname*, and an index *n*. The entry appears on the *n*th *dayname* after the first day of *month*, where *dayname*=0 means Sunday, 1 means Monday, and so on. If *n* is negative it counts backward from the end of *month*. The value of *month* can be a list of months, a single month, or `t` to specify all months. You can also use an optional parameter *day* to specify the *n*th *dayname* on or after/before *day* of *month*; the value of *day* defaults to 1 if *n* is positive and to the last day of *month* if *n* is negative. For example,

```
%%(diary-float t 1 -1) Pay rent
```

causes the fancy diary to show

```
Pay rent
```

on the last Monday of every month.

The generality of sexp diary entries lets you specify any diary entry that you can describe algorithmically. A sexp diary entry contains an expression that computes whether the entry applies to any given date. If its value is non-`nil`, the entry applies to that date; otherwise, it does not. The expression can use the variable `date` to find the date being considered; its value is a list (*month day year*) that refers to the Gregorian calendar.

The sexp diary entry applies to a date when the expression's value is non-`nil`, but some values have more specific meanings. If the value is a string, that string is a description of the event which occurs on that date. The value can also have the form `(mark . string)`; then *mark* specifies how to mark the date in the calendar, and *string* is the description of the event. If *mark* is a single-character string, that character appears next to the date in the calendar. If *mark* is a face name, the date is displayed in that face. If *mark* is `nil`, that specifies no particular highlighting for the date.

Suppose you get paid on the 21st of the month if it is a weekday, and on the Friday before if the 21st is on a weekend. Here is how to write a sexp diary entry that matches those dates:

```
&%%(let ((dayname (calendar-day-of-week date))
         (day (cadr date)))
      (or (and (= day 21) (memq dayname '(1 2 3 4 5)))
          (and (memq day '(19 20)) (= dayname 5)))
         ) Pay check deposited
```

The following sexp diary entries take advantage of the ability (in the fancy diary display) to concoct diary entries whose text varies based on the date:

`%%(diary-sunrise-sunset)`
: Make a diary entry for today's local times of sunrise and sunset.

`%%(diary-lunar-phases)`
: Make a diary entry for the phases (quarters) of the moon.

`%%(diary-day-of-year)`
: Make a diary entry with today's day number in the current year and the number of days remaining in the current year.

`%%(diary-iso-date)`
: Make a diary entry with today's equivalent ISO commercial date.

`%%(diary-julian-date)`
: Make a diary entry with today's equivalent Julian calendar date.

`%%(diary-astro-day-number)`
: Make a diary entry with today's equivalent astronomical (Julian) day number.

`%%(diary-bahai-date)`
: Make a diary entry with today's equivalent Bahá'í calendar date.

`%%(diary-chinese-date)`
: Make a diary entry with today's equivalent Chinese calendar date.

`%%(diary-coptic-date)`
: Make a diary entry with today's equivalent Coptic calendar date.

`%%(diary-ethiopic-date)`
: Make a diary entry with today's equivalent Ethiopic calendar date.

`%%(diary-french-date)`
: Make a diary entry with today's equivalent date on the French Revolutionary calendar.

`%%(diary-hebrew-date)`
: Make a diary entry with today's equivalent Hebrew calendar date.

`%%(diary-islamic-date)`
: Make a diary entry with today's equivalent Islamic calendar date.

`%%(diary-mayan-date)`
: Make a diary entry with today's equivalent Mayan calendar date.

`%%(diary-persian-date)`
: Make a diary entry with today's equivalent Persian calendar date.

For example, including the diary entry

```
&%%(diary-hebrew-date)
```

causes every day's diary display to contain the equivalent date on the Hebrew calendar, if you are using the fancy diary display. (With simple diary display, the literal line '`&%%(diary-hebrew-date)`' appears in the diary for any date.)

This function has been used to construct certain standard Hebrew sexp diary entries:

`%%(diary-hebrew-rosh-hodesh)`
: Make a diary entry that tells the occurrence and ritual announcement of each new Hebrew month.

`%%(diary-hebrew-parasha)`
: Make a Saturday diary entry that tells the weekly synagogue scripture reading.

`%%(diary-hebrew-sabbath-candles)`
: Make a Friday diary entry that tells the *local time* of Sabbath candle lighting.

`%%(diary-hebrew-omer)`
: Make a diary entry that gives the omer count, when appropriate.

`%%(diary-hebrew-yahrzeit` *`month day year`*`)` *`name`*
: Make a diary entry marking the anniversary of a date of death. The date is the *Gregorian* (civil) date of death. The diary entry appears on the proper Hebrew calendar anniversary and on the day before. (The order of the parameters changes according to the calendar date style; for example in the European style to *day*, *month*, *year*.)

`%%(diary-hebrew-birthday` *`month day year`*`)`
: Make a diary entry for a birthday on the Hebrew calendar.

All the functions documented above take an optional argument *mark* which specifies how to mark the date in the calendar display. If one of these functions decides that it applies to a certain date, it returns a value that contains *mark*, as described above.

Merging Files with Emerge

It's not unusual for programmers to get their signals crossed and modify the same program in two different directions. To recover from this confusion, you need to merge the two versions. Emerge makes this easier. For other ways to compare files, see Section "Comparing Files" in *the Emacs Manual*, and Section "Ediff" in *The Ediff Manual*.

Overview of Emerge

To start Emerge, run one of these four commands:

`M-x emerge-files`
: Merge two specified files.

`M-x emerge-files-with-ancestor`
: Merge two specified files, with reference to a common ancestor.

`M-x emerge-buffers`
: Merge two buffers.

`M-x emerge-buffers-with-ancestor`
: Merge two buffers with reference to a common ancestor in a third buffer.

The Emerge commands compare two files or buffers, and display the comparison in three buffers: one for each input text (the *A buffer* and the *B buffer*), and one (the *merge buffer*) where merging takes place. The merge buffer shows the full merged text, not just the differences. Wherever the two input texts differ, you can choose which one of them to include in the merge buffer.

The Emerge commands that take input from existing buffers use only the accessible portions of those buffers, if they are narrowed. See Section "Narrowing" in *the Emacs Manual*.

If a common ancestor version is available, from which the two texts to be merged were both derived, Emerge can use it to guess which alternative is right. Wherever one current version agrees with the ancestor, Emerge presumes that the other current version is a deliberate change which should be kept in the merged version. Use the '`with-ancestor`' commands if you want to specify a common ancestor text. These commands read three file or buffer names—variant A, variant B, and the common ancestor.

After the comparison is done and the buffers are prepared, the interactive merging starts. You control the merging by typing special *merge commands* in the merge buffer (see [Merge Commands], page 22). For each run of differences between the input texts, you can choose which one of them to keep, or edit them both together.

The merge buffer uses a special major mode, Emerge mode, with commands for making these choices. But you can also edit the buffer with ordinary Emacs commands.

At any given time, the attention of Emerge is focused on one particular difference, called the *selected* difference. This difference is marked off in the three buffers like this:

```
vvvvvvvvvvvvvvvvvvvv
text that differs
^^^^^^^^^^^^^^^^^^^^
```

Emerge numbers all the differences sequentially and the mode line always shows the number of the selected difference.

Normally, the merge buffer starts out with the A version of the text. But when the A version of a difference agrees with the common ancestor, then the B version is initially preferred for that difference.

Emerge leaves the merged text in the merge buffer when you exit. At that point, you can save it in a file with *C-x C-w*. If you give a numeric argument to `emerge-files` or `emerge-files-with-ancestor`, it reads the name of the output file using the minibuffer. (This is the last file name those commands read.) Then exiting from Emerge saves the merged text in the output file.

Normally, Emerge commands save the output buffer in its file when you exit. If you abort Emerge with *C-]*, the Emerge command does not save the output buffer, but you can save it yourself if you wish.

Submodes of Emerge

You can choose between two modes for giving merge commands: Fast mode and Edit mode. In Fast mode, basic merge commands are single characters, but ordinary Emacs commands are disabled. This is convenient if you use only merge commands. In Edit mode, all merge commands start with the prefix key *C-c C-c*, and the normal Emacs commands are also available. This allows editing the merge buffer, but slows down Emerge operations.

Use *e* to switch to Edit mode, and *C-c C-c f* to switch to Fast mode. The mode line indicates Edit and Fast modes with '`E`' and '`F`'.

Emerge has two additional submodes that affect how particular merge commands work: Auto Advance mode and Skip Prefers mode.

If Auto Advance mode is in effect, the *a* and *b* commands advance to the next difference. This lets you go through the merge faster as long as you simply choose one of the alternatives from the input. The mode line indicates Auto Advance mode with '`A`'.

If Skip Prefers mode is in effect, the *n* and *p* commands skip over differences in states "prefer-A" and "prefer-B" (see [State of Difference], page 21). Thus you see only differences for which neither version is presumed correct. The mode line indicates Skip Prefers mode with '`S`'. This mode is only relevant when there is an ancestor.

Use the command *s a* (`emerge-auto-advance`) to set or clear Auto Advance mode. Use *s s* (`emerge-skip-prefers`) to set or clear Skip Prefers mode. These commands turn on the mode with a positive argument, turn it off with a negative or zero argument, and toggle the mode with no argument.

State of a Difference

In the merge buffer, a difference is marked with lines of '`v`' and '`^`' characters. Each difference has one of these seven states:

A
: The difference is showing the A version. The *a* command always produces this state; the mode line indicates it with '`A`'.

B
: The difference is showing the B version. The *b* command always produces this state; the mode line indicates it with '`B`'.

default-A
default-B
: The difference is showing the A or the B state by default, because you haven't made a choice. All differences start in the default-A state (and thus the merge buffer is a copy of the A buffer), except those for which one alternative is preferred (see below).

  When you select a difference, its state changes from default-A or default-B to plain A or B. Thus, the selected difference never has state default-A or default-B, and these states are never displayed in the mode line.

  The command `d a` chooses default-A as the default state, and `d b` chooses default-B. This chosen default applies to all differences that you have never selected and for which no alternative is preferred. If you are moving through the merge sequentially, the differences you haven't selected are those following the selected one. Thus, while moving sequentially, you can effectively make the A version the default for some sections of the merge buffer and the B version the default for others by using `d a` and `d b` between sections.

prefer-A
prefer-B
: The difference is showing the A or B state because it is *preferred*. This means that you haven't made an explicit choice, but one alternative seems likely to be right because the other alternative agrees with the common ancestor. Thus, where the A buffer agrees with the common ancestor, the B version is preferred, because chances are it is the one that was actually changed.

  These two states are displayed in the mode line as '`A*`' and '`B*`'.

combined
: The difference is showing a combination of the A and B states, as a result of the `x c` or `x C` commands.

  Once a difference is in this state, the `a` and `b` commands don't do anything to it unless you give them a numeric argument.

  The mode line displays this state as '`comb`'.

Merge Commands

Here are the Merge commands for Fast mode; in Edit mode, precede them with `C-c C-c`:

`p`
: Select the previous difference.

`n`
: Select the next difference.

`a`
: Choose the A version of this difference.

`b`
: Choose the B version of this difference.

`C-u n j`
: Select difference number *n*.

`.`
: Select the difference containing point.

`q`
: Quit—finish the merge.

`C-]`
: Abort—exit merging and do not save the output.

`f`
: Go into Fast mode. (In Edit mode, this is actually `C-c C-c f`.)

`e`
: Go into Edit mode.

`l` Recenter (like `C-l`) all three windows. With an argument, reestablish the default three-window display.

`-` Specify part of a prefix numeric argument.

`digit` Also specify part of a prefix numeric argument.

`d a` Choose the A version as the default from here down in the merge buffer.

`d b` Choose the B version as the default from here down in the merge buffer.

`c a` Copy the A version of this difference into the kill ring.

`c b` Copy the B version of this difference into the kill ring.

`i a` Insert the A version of this difference at point.

`i b` Insert the B version of this difference at point.

`m` Put point and mark around the difference.

`^` Scroll all three windows down (like `M-v`).

`v` Scroll all three windows up (like `C-v`).

`<` Scroll all three windows left (like `C-x <`).

`>` Scroll all three windows right (like `C-x >`).

`|` Reset horizontal scroll on all three windows.

`x 1` Shrink the merge window to one line. (Use `C-u l` to restore it to full size.)

`x c` Combine the two versions of this difference (see [Combining in Emerge], page 24).

`x f` Show the names of the files/buffers Emerge is operating on, in a Help window. (Use `C-u l` to restore windows.)

`x j` Join this difference with the following one. (`C-u x j` joins this difference with the previous one.)

`x s` Split this difference into two differences. Before you use this command, position point in each of the three buffers at the place where you want to split the difference.

`x t` Trim identical lines off the top and bottom of the difference. Such lines occur when the A and B versions are identical but differ from the ancestor version.

Exiting Emerge

The `q` command (`emerge-quit`) finishes the merge, storing the results into the output file if you specified one. It restores the A and B buffers to their proper contents, or kills them if they were created by Emerge and you haven't changed them. It also disables the Emerge commands in the merge buffer, since executing them later could damage the contents of the various buffers.

`C-]` aborts the merge. This means exiting without writing the output file. If you didn't specify an output file, then there is no real difference between aborting and finishing the merge.

If the Emerge command was called from another Lisp program, then its return value is `t` for successful completion, or `nil` if you abort.

Combining the Two Versions

Sometimes you want to keep *both* alternatives for a particular difference. To do this, use *x c*, which edits the merge buffer like this:

```
#ifdef NEW
version from B buffer
#else /* not NEW */
version from A buffer
#endif /* not NEW */
```

While this example shows C preprocessor conditionals delimiting the two alternative versions, you can specify the strings to use by setting the variable `emerge-combine-versions-template` to a string of your choice. In the string, '`%a`' says where to put version A, and '`%b`' says where to put version B. The default setting, which produces the results shown above, looks like this:

```
"#ifdef NEW\n%b#else /* not NEW */\n%a#endif /* not NEW */\n"
```

Fine Points of Emerge

During the merge, you mustn't try to edit the A and B buffers yourself. Emerge modifies them temporarily, but ultimately puts them back the way they were.

You can have any number of merges going at once—just don't use any one buffer as input to more than one merge at once, since the temporary changes made in these buffers would get in each other's way.

Starting Emerge can take a long time because it needs to compare the files fully. Emacs can't do anything else until `diff` finishes. Perhaps in the future someone will change Emerge to do the comparison in the background when the input files are large—then you could keep on doing other things with Emacs until Emerge is ready to accept commands.

After setting up the merge, Emerge runs the hook `emerge-startup-hook`. See Section "Hooks" in *the Emacs Manual*.

Advanced VC Usage

Commonly used features of Emacs's version control (VC) support are described in the main Emacs manual (see Section "Version Control" in *the Emacs Manual*). This chapter describes more advanced VC usage.

Miscellaneous Commands and Features of VC

This section explains the less-frequently-used features of VC.

Change Logs and VC

If you use RCS or CVS for a program with a `ChangeLog` file (see Section "Change Log" in *the Emacs Manual*), you can generate change log entries from the version control log entries of previous commits.

Note that this only works with RCS or CVS. This procedure would be particularly incorrect on a modern changeset-based version control system, where changes to the `ChangeLog` file would normally be committed as part of a changeset. In that case, you should write the change log entries first, then pull them into the '`*vc-log*`' buffer when you commit (see Section "Log Buffer" in *the Emacs Manual*).

C-x v a Visit the current directory's `ChangeLog` file and, for registered files in that directory, create new entries for versions committed since the most recent change log entry (`vc-update-change-log`).

C-u C-x v a
As above, but only find entries for the current buffer's file.

For example, suppose the first line of `ChangeLog` is dated 1999-04-10, and that the only check-in since then was by Nathaniel Bowditch to `rcs2log` on 1999-05-22 with log entry '`Ignore log messages that start with '#'.`'. Then *C-x v a* inserts this `ChangeLog` entry:

```
1999-05-22  Nathaniel Bowditch  <nat@apn.org>

        * rcs2log: Ignore log messages that start with '#'.
```

If the version control log entry specifies a function name (in parenthesis at the beginning of a line), that is reflected in the `ChangeLog` entry. For example, if a log entry for `vc.el` is '`(vc-do-command): Check call-process status.`', the `ChangeLog` entry is:

```
1999-05-06  Nathaniel Bowditch  <nat@apn.org>

        * vc.el (vc-do-command): Check call-process status.
```

When *C-x v a* adds several change log entries at once, it groups related log entries together if they all are checked in by the same author at nearly the same time. If the log entries for several such files all have the same text, it coalesces them into a single entry.

Deleting and Renaming Version-Controlled Files

M-x vc-delete-file
Prompt for a file name, delete the file from the working tree, and schedule the deletion for committing.

`M-x vc-rename-file`
: Prompt for two file names, *old* and *new*, rename them in the working tree, and schedule the renaming for committing. The *old* file defaults to the current buffer's file name if it is under VC.

If you wish to delete a version-controlled file, use the command `M-x vc-delete-file`. This prompts for the file name, and deletes it via the version control system. The file is removed from the working tree, and in the VC Directory buffer (see Section "VC Directory Mode" in *the Emacs Manual*), it is displayed with the '`removed`' status. When you commit it, the deletion takes effect in the repository.

To rename a version-controlled file, type `M-x vc-rename-file`. This prompts for two arguments: the name of the file you wish to rename, and the new name; then it performs the renaming via the version control system. The renaming takes effect immediately in the working tree, and takes effect in the repository when you commit the renamed file.

On modern version control systems that have built-in support for renaming, the renamed file retains the full change history of the original file. On CVS and older version control systems, the `vc-rename-file` command actually works by creating a copy of the old file under the new name, registering it, and deleting the old file. In this case, the change history is not preserved.

Revision Tags

Most version control systems allow you to apply a *revision tag* to a specific version of a version-controlled tree. On modern changeset-based version control systems, a revision tag is simply a symbolic name for a particular revision. On older file-based systems like CVS, each tag is added to the entire set of version-controlled files, allowing them to be handled as a unit. Revision tags are commonly used to identify releases that are distributed to users.

There are two basic commands for tags; one makes a tag with a given name, the other retrieves a named tag.

`C-x v s` *name* `RET`
: Define the working revision of every registered file in or under the current directory as a tag named *name* (`vc-create-tag`).

`C-x v r` *name* `RET`
: For all registered files at or below the current directory level, retrieve the tagged revision *name*. This command will switch to a branch if *name* is a branch name and your VCS distinguishes branches from tags. (`vc-retrieve-tag`).

: This command reports an error if any files are locked at or below the current directory, without changing anything; this is to avoid overwriting work in progress.

You can give a tag or branch name as an argument to `C-x v =` or `C-x v ~` (see Section "Old Revisions" in *the Emacs Manual*). Thus, you can use it to compare a tagged version against the current files, or two tagged versions against each other.

On SCCS, VC implements tags itself; these tags are visible only through VC. Most later systems (including CVS, Subversion, bzr, git, and hg) have a native tag facility, and VC uses it where available; those tags will be visible even when you bypass VC.

In file-based version control systems, when you rename a registered file you need to rename its master along with it; the command `vc-rename-file` will do this automatically (see Section "VC Delete/Rename" in *the Emacs Manual*). If you are using SCCS, you must also update the records of the tag, to mention the file by its new name (`vc-rename-file` does this, too). An old tag that refers to a master file that no longer exists under the recorded name is invalid; VC can no longer retrieve it. It would be beyond the scope of this manual to explain enough about RCS and SCCS to explain how to update the tags by hand. Using `vc-rename-file` makes the tag remain valid for retrieval, but it does not solve all problems. For example, some of the files in your program probably refer to others by name. At the very least, the makefile probably mentions the file that you renamed. If you retrieve an old tag, the renamed file is retrieved under its new name, which is not the name that the makefile expects. So the program won't really work as retrieved.

Inserting Version Control Headers

On Subversion, CVS, RCS, and SCCS, you can put certain special strings called *version headers* into a work file. When the file is committed, the version control system automatically puts the revision number, the name of the user who made the commit, and other relevant information into the version header.

VC does not normally use the information in the version headers. As an exception, when using RCS, Emacs uses the version header, if there is one, to determine the file version, since it is often more reliable than the RCS master file. To inhibit using the version header this way, change the variable `vc-consult-headers` to `nil`. VC then always uses the file permissions (if it is supposed to trust them), or else checks the master file.

To insert a suitable header string into the current buffer, use the command *M-x vc-insert-headers*. This command works only on Subversion, CVS, RCS, and SCCS. The variable `vc-`*backend*`-header` contains the list of keywords to insert into the version header; for instance, CVS uses `vc-cvs-header`, whose default value is `'("\$Id\$")`. (The extra backslashes prevent the string constant from being interpreted as a header, if the Emacs Lisp file defining it is maintained with version control.) The `vc-insert-headers` command inserts each keyword in the list on a new line at point, surrounded by tabs, and inside comment delimiters if necessary.

The variable `vc-static-header-alist` specifies further strings to add based on the name of the buffer. Its value should be a list of elements of the form `(`*regexp* `.` *format*`)`. Whenever *regexp* matches the buffer name, *format* is also inserted as part of the version header. A '`%s`' in *format* is replaced with the file's version control type.

Editing VC Commands

You can use the *C-x v !* (`vc-edit-next-command`) prefix command to edit the shell command line that VC is about to run. This is primarily intended to make it possible to add optional command-line arguments to VCS commands without unnecessary complications of the VC command set and its interfaces with the backend.

For example, Git can produce logs of more than one branch, but *C-x v b l* (`vc-print-branch-log`) prompts for the name of just one branch. To obtain a log of more than one branch, you can type *C-x v ! C-x v b l* and then append the names of additional branches to the end of the '`git log`' command that VC is about to run.

Preparing Patches

When collaborating on projects it is common to send patches via email, to share changes. You can do this using VC with the `vc-prepare-patch` command. This will prompt you for the revisions you wish to share, and which destination email address(es) to use. Separate the revisions using the value of *crm-separator*, commas by default. The command will then prepare those revisions using your MUA (Mail User Agent) for you to review and send.

When invoked interactively in a Log View buffer with marked revisions, those marked revisions will be used.

Depending on the value of the user option `vc-prepare-patches-separately`, `vc-prepare-patch` will generate one or more messages. The default value `t` means prepare and display a message for each revision, one after another. A value of `nil` means to generate a single message with all patches attached in the body.

If you expect to contribute patches on a regular basis, you can set the user option `vc-default-patch-addressee` to the address(es) you wish to use. This will be used as the default value when invoking `vc-prepare-patch`. Project maintainers may consider setting this as a directory local variable (see ⟨undefined⟩ [Directory Variables], page ⟨undefined⟩).

Customizing VC

The variable `vc-handled-backends` determines which version control systems VC should handle. The default value is `(RCS CVS SVN SCCS SRC Bzr Git Hg)`, so it contains all the version systems that are currently supported. If you want VC to ignore one or more of these systems, exclude its name from the list. To disable VC entirely, set this variable to `nil`.

The order of systems in the list is significant: when you visit a file registered in more than one system, VC uses the system that comes first in `vc-handled-backends` by default. The order is also significant when you register a file for the first time (see Section “Registering” in *the Emacs Manual*).

General Options

Emacs normally does not save backup files for source files that are maintained with version control. If you want to make backup files even for files that use version control, set the variable `vc-make-backup-files` to a non-`nil` value.

Editing a version-controlled file through a symbolic link may cause unexpected results, if you are unaware that the underlying file is version-controlled. The variable `vc-follow-symlinks` controls what Emacs does if you try to visit a symbolic link pointing to a version-controlled file. If the value is `ask` (the default), Emacs asks for confirmation. If it is `nil`, Emacs just displays a warning message. If it is `t`, Emacs automatically follows the link and visits the real file instead.

If `vc-suppress-confirm` is non-`nil`, then *`C-x v v`* and *`C-x v i`* can save the current buffer without asking, and *`C-x v u`* also operates without asking for confirmation.

VC mode does much of its work by running the shell commands for the appropriate version control system. If `vc-command-messages` is non-`nil`, VC displays messages to indicate which shell commands it runs, and additional messages when the commands finish.

Options for RCS and SCCS

By default, RCS uses locking to coordinate the activities of several users, but there is a mode called *non-strict locking* in which you can check-in changes without locking the file first. Use '`rcs -U`' to switch to non-strict locking for a particular file, see the `rcs` manual page for details.

When deducing the version control state of an RCS file, VC first looks for an RCS version header string in the file (see [Version Headers], page 27). If there is no header string, VC normally looks at the file permissions of the work file; this is fast. But there might be situations when the file permissions cannot be trusted. In this case the master file has to be consulted, which is rather expensive. Also the master file can only tell you *if* there's any lock on the file, but not whether your work file really contains that locked version.

You can tell VC not to use version headers to determine the file status by setting `vc-consult-headers` to `nil`. VC then always uses the file permissions (if it is supposed to trust them), or else checks the master file.

VC determines the version control state of files under SCCS much as with RCS. It does not consider SCCS version headers, though. Thus, the variable `vc-consult-headers` does not affect SCCS use.

Options specific for CVS

You can specify additional command line options to pass to all CVS operations in the variable `vc-cvs-global-switches`. These switches are inserted immediately after the `cvs` command, before the name of the operation to invoke.

When using a CVS repository on a remote machine, VC can try keeping network interactions to a minimum. This is controlled by the variable `vc-cvs-stay-local`. If `vc-cvs-stay-local` is `only-file` (the default), VC determines the version control status of each file using only the entry in the local CVS subdirectory and the information returned by previous CVS commands. As a consequence, if you have modified a file and somebody else has checked in other changes, you will not be notified of the conflict until you try to commit.

If you change `vc-cvs-stay-local` to `nil`, VC queries the remote repository *before* it decides what to do in `vc-next-action` (*C-x v v*), just as it does for local repositories.

You can also set `vc-cvs-stay-local` to a regular expression that is matched against the repository host name; VC then stays local only for repositories from hosts that match the pattern.

When using a remote repository, Emacs normally makes *automatic version backups* of the original versions of each edited file. These local backups are made whenever you save the first changes to a file, and they are removed after you commit your changes to the repository. (Note that these are not the same as ordinary Emacs backup files; see Section "Backup" in *the Emacs Manual*.) Commands like *C-x v =* and *C-x v u* make use of automatic version backups, if possible, to avoid having to access the network.

Setting `vc-cvs-stay-local` to `nil` disables the making of automatic version backups.

Automatic version backups have names of the form *`file.~version.~`*. This is similar to the name that *C-x v ~* saves old versions to (see Section "Old Revisions" in *the Emacs Manual*), except for the additional dot ('`.`') after the version. The relevant VC commands can use both kinds of version backups. The main difference is that the manual version backups made by `C-x v ~` are not deleted automatically when you commit.

CVS does not use locking by default, but there are ways to enable locking-like behavior using its `CVSREAD` or *watch* feature; see the CVS documentation for details. If that case, you can use `C-x v v` in Emacs to toggle locking, as you would for a locking-based version control system (see Section "VC With A Locking VCS" in *the Emacs Manual*).

Fortran Mode

Fortran mode is meant for editing fixed form (and also tab format) source code (normally Fortran 77). For editing more modern free-form source code (Fortran 90, 95, 2003, 2008), use F90 mode (`f90-mode`). Emacs normally uses Fortran mode for files with extension '`.f`', '`.F`' or '`.for`', and F90 mode for the extensions '`.f90`', '`.f95`', '`.f03`' and '`.f08`'. Customize `auto-mode-alist` to add more extensions. GNU Fortran supports both free and fixed form. This manual mainly documents Fortran mode, but the corresponding F90 mode features are mentioned when relevant.

Fortran mode provides special motion commands for Fortran statements and subprograms, and indentation commands that understand Fortran conventions of nesting, line numbers and continuation statements. Fortran mode has support for Auto Fill mode that breaks long lines into proper Fortran continuation lines. Fortran mode also supports Hideshow minor mode (see Section "Hideshow" in *the Emacs Manual*), and Imenu (see Section "Imenu" in *the Emacs Manual*).

Special commands for comments are provided because Fortran comments are unlike those of other languages. Built-in abbrevs optionally save typing when you insert Fortran keywords.

Use *M-x fortran-mode* to switch to this major mode. This command runs the hook `fortran-mode-hook`. See Section "Hooks" in *the Emacs Manual*.

Motion Commands

In addition to the normal commands for moving by and operating on defuns (Fortran subprograms—functions and subroutines, as well as modules for F90 mode, using the commands `fortran-end-of-subprogram` and `fortran-beginning-of-subprogram`), Fortran mode provides special commands to move by statements and other program units.

C-c C-n
: Move to the beginning of the next statement (`fortran-next-statement/f90-next-statement`).

C-c C-p
: Move to the beginning of the previous statement (`fortran-previous-statement/f90-previous-statement`). If there is no previous statement (i.e., if called from the first statement in the buffer), move to the start of the buffer.

C-c C-e
: Move point forward to the start of the next code block, or the end of the current one, whichever comes first (`f90-next-block`). A code block is a subroutine, `if`–`endif` statement, and so forth. This command exists for F90 mode only, not Fortran mode. With a numeric argument, it moves forward that many blocks.

C-c C-a
: Move point backward to the previous block (`f90-previous-block`). This is like `f90-next-block`, but moves backwards.

C-M-n
: Move to the end of the current code block (`fortran-end-of-block/f90-end-of-block`). With a numeric argument, move forward that number of blocks. The mark is set before moving point. The F90 mode version of this command checks for consistency of block types and labels (if present), but it does not check the outermost block since that may be incomplete.

C-M-p Move to the start of the current code block (`fortran-beginning-of-block/f90-beginning-of-block`). This is like `fortran-end-of-block`, but moves backwards.

The commands `fortran-beginning-of-subprogram` and `fortran-end-of-subprogram` move to the start or end of the current subprogram, respectively. The commands `fortran-mark-do` and `fortran-mark-if` mark the end of the current `do` or `if` block, and move point to the start.

Fortran Indentation

Special commands and features are needed for indenting fixed (or tab) form Fortran code in order to make sure various syntactic entities (line numbers, comment line indicators and continuation line flags) appear in the required columns.

Fortran Indentation and Filling Commands

C-M-j Break the current line at point and set up a continuation line (`fortran-split-line`).

*M-^* Join this line to the previous line (`fortran-join-line`).

C-M-q Indent all the lines of the subprogram that point is in (`fortran-indent-subprogram`).

M-q Fill a comment block or statement (using `fortran-fill-paragraph` or `fortran-fill-statement`).

The key *C-M-q* runs `fortran-indent-subprogram`, a command to reindent all the lines of the Fortran subprogram (function or subroutine) containing point.

The key *C-M-j* runs `fortran-split-line`, which splits a line in the appropriate fashion for Fortran. In a non-comment line, the second half becomes a continuation line and is indented accordingly. In a comment line, both halves become separate comment lines.

*M-^* or *C-c C-d* run the command `fortran-join-line`, which joins a continuation line back to the previous line, roughly as the inverse of `fortran-split-line`. The point must be on a continuation line when this command is invoked.

M-q in Fortran mode fills the comment block or statement that point is in. This removes any excess statement continuations.

Continuation Lines

Most Fortran 77 compilers allow two ways of writing continuation lines. If the first non-space character on a line is in column 5, then that line is a continuation of the previous line. We call this *fixed form*. (In GNU Emacs we always count columns from 0; but note that the Fortran standard counts from 1. You can customize the variable `column-number-indicator-zero-based` to make the column display Fortran-like; see Section "Optional Mode Line" in *the Emacs Manual*.) The variable `fortran-continuation-string` specifies what character to put in column 5. A line that starts with a tab character followed by any digit except '`0`' is also a continuation line. We call this style of continuation *tab format*. (Fortran 90 introduced free-form continuation lines.)

Fortran mode can use either style of continuation line. When you enter Fortran mode, it tries to deduce the proper continuation style automatically from the buffer contents. It does this by scanning up to `fortran-analyze-depth` (default 100) lines from the start of the buffer. The first line that begins with either a tab character or six spaces determines the choice. If the scan fails (for example, if the buffer is new and therefore empty), the value of `fortran-tab-mode-default` (`nil` for fixed form, and non-`nil` for tab format) is used. '`/t`' (`fortran-tab-mode-string`) in the mode line indicates tab format is selected. Fortran mode sets the value of `indent-tabs-mode` accordingly.

If the text on a line starts with the Fortran continuation marker '`$`', or if it begins with any non-whitespace character in column 5, Fortran mode treats it as a continuation line. When you indent a continuation line with `TAB`, it converts the line to the current continuation style. When you split a Fortran statement with *`C-M-j`*, the continuation marker on the newline is created according to the continuation style.

The setting of continuation style affects several other aspects of editing in Fortran mode. In fixed form mode, the minimum column number for the body of a statement is 6. Lines inside of Fortran blocks that are indented to larger column numbers must use only the space character for whitespace. In tab format mode, the minimum column number for the statement body is 8, and the whitespace before column 8 must consist of one tab character.

Line Numbers

If a number is the first non-whitespace in the line, Fortran indentation assumes it is a line number and moves it to columns 0 through 4. (Columns always count from 0 in Emacs, but setting `column-number-indicator-zero-based` to `nil` can change that, see Section "Optional Mode Line" in *the Emacs Manual*.)

Line numbers of four digits or less are normally indented one space. The variable `fortran-line-number-indent` controls this; it specifies the maximum indentation a line number can have. The default value of the variable is 1. Fortran mode tries to prevent line number digits passing column 4, reducing the indentation below the specified maximum if necessary. If `fortran-line-number-indent` has the value 5, line numbers are right-justified to end in column 4.

Simply inserting a line number is enough to indent it according to these rules. As each digit is inserted, the indentation is recomputed. To turn off this feature, set the variable `fortran-electric-line-number` to `nil`.

Syntactic Conventions

Fortran mode assumes that you follow certain conventions that simplify the task of understanding a Fortran program well enough to indent it properly:

- Two nested '`do`' loops never share a '`continue`' statement.
- Fortran keywords such as '`if`', '`else`', '`then`', '`do`' and others are written without embedded whitespace or line breaks.

  Fortran compilers generally ignore whitespace outside of string constants, but Fortran mode does not recognize these keywords if they are not contiguous. Constructs such as '`else if`' or '`end do`' are acceptable, but the second word should be on the same line as the first and not on a continuation line.

If you fail to follow these conventions, the indentation commands may indent some lines unaesthetically. However, a correct Fortran program retains its meaning when reindented even if the conventions are not followed.

Variables for Fortran Indentation

Several additional variables control how Fortran indentation works:

`fortran-do-indent`
: Extra indentation within each level of '`do`' statement (default 3).

`fortran-if-indent`
: Extra indentation within each level of '`if`', '`select case`', or '`where`' statements (default 3).

`fortran-structure-indent`
: Extra indentation within each level of '`structure`', '`union`', '`map`', or '`interface`' statements (default 3).

`fortran-continuation-indent`
: Extra indentation for bodies of continuation lines (default 5).

`fortran-check-all-num-for-matching-do`
: In Fortran 77, a numbered '`do`' statement is terminated by any statement with a matching line number. It is common (but not compulsory) to use a '`continue`' statement for this purpose. If this variable has a non-`nil` value, indenting any numbered statement must check for a '`do`' that ends there. If you always end '`do`' statements with a '`continue`' line (or if you use the more modern '`enddo`'), then you can speed up indentation by setting this variable to `nil` (the default).

`fortran-blink-matching-if`
: If this is `t`, indenting an '`endif`' (or '`enddo`') statement moves the cursor momentarily to the matching '`if`' (or '`do`') statement to show where it is. The default is `nil`.

`fortran-minimum-statement-indent-fixed`
: Minimum indentation for Fortran statements when using fixed form continuation line style. Statement bodies are never indented by less than this. The default is 6.

`fortran-minimum-statement-indent-tab`
: Minimum indentation for Fortran statements for tab format continuation line style. Statement bodies are never indented by less than this. The default is 8.

The following section describes the variables controlling the indentation of comments.

Fortran Comments

The usual Emacs comment commands assume that a comment can follow a line of code. In Fortran 77, the standard comment syntax requires an entire line to be just a comment. Therefore, Fortran mode replaces the standard Emacs comment commands and defines some new variables.

Fortran mode can also handle the Fortran 90 comment syntax where comments start with '`!`' and can follow other text. Because only some Fortran 77 compilers accept this syntax, Fortran mode will not insert such comments unless you have said in advance to do so. To do this, set the variable `fortran-comment-line-start` to '`"!"`'. If you use an unusual value, you may need to change `fortran-comment-line-start-skip`.

M-; Align comment or insert new comment (`comment-dwim`).

C-x ; Applies to nonstandard '`!`' comments only (`comment-set-column`).

C-c ; Turn all lines of the region into comments, or (with argument) turn them back into real code (`fortran-comment-region`).

M-; in Fortran mode runs the standard `comment-dwim`. This recognizes any kind of existing comment and aligns its text appropriately; if there is no existing comment, a comment is inserted and aligned. Inserting and aligning comments are not the same in Fortran mode as in other modes.

When a new comment must be inserted, if the current line is blank, a full-line comment is inserted. On a non-blank line, a nonstandard '`!`' comment is inserted if you have said you want to use them. Otherwise, a full-line comment is inserted on a new line before the current line.

Nonstandard '`!`' comments are aligned like comments in other languages, but full-line comments are different. In a standard full-line comment, the comment delimiter itself must always appear in column zero. What can be aligned is the text within the comment. You can choose from three styles of alignment by setting the variable `fortran-comment-indent-style` to one of these values:

`fixed` Align the text at a fixed column, which is the sum of `fortran-comment-line-extra-indent` and the minimum statement indentation. This is the default.

The minimum indentation is `fortran-minimum-statement-indent-tab` for tab format continuation line style and `fortran-minimum-statement-indent-fixed` for fixed form style.

`relative` Align the text as if it were a line of code, but with an additional `fortran-comment-line-extra-indent` columns of indentation.

`nil` Don't move text in full-line comments automatically.

In addition, you can specify the character to be used to indent within full-line comments by setting the variable `fortran-comment-indent-char` to the single-character string you want to use.

Compiler directive lines, or preprocessor lines, have much the same appearance as comment lines. It is important, though, that such lines never be indented at all, no matter what the value of `fortran-comment-indent-style`. The variable `fortran-directive-re` is a regular expression that specifies which lines are directives. Matching lines are never indented, and receive distinctive font-locking.

The normal Emacs comment command *C-x ;* (`comment-set-column`) has not been redefined. If you use '`!`' comments, this command can be used with them. Otherwise, it is useless in Fortran mode.

The command *C-c ;* (`fortran-comment-region`) turns all the lines of the region into comments by inserting the string '`c$$$`' at the front of each one. With a numeric argument, it turns the region back into live code by deleting '`c$$$`' from the front of each line in it. The string used for these comments can be controlled by setting the variable `fortran-comment-region`. Note that here we have an example of a command and a variable with the same name; these two uses of the name never conflict because in Lisp and in Emacs it is always clear from the context which one is meant.

Auto Fill in Fortran Mode

Fortran mode has specialized support for Auto Fill mode, which is a minor mode that automatically splits statements as you insert them when they become too wide. Splitting a statement involves making continuation lines using `fortran-continuation-string` (see [ForIndent Cont], page 32). This splitting happens when you type `SPC`, `RET`, or `TAB`, and also in the Fortran indentation commands. You activate Auto Fill in Fortran mode in the normal way. See Section "Auto Fill" in *the Emacs Manual*.

Auto Fill breaks lines at spaces or delimiters when the lines get longer than the desired width (the value of `fill-column`). The delimiters (besides whitespace) that Auto Fill can break at are '`+`', '`-`', '`/`', '`*`', '`=`', '`<`', '`>`', and '`,`'. The line break comes after the delimiter if the variable `fortran-break-before-delimiters` is `nil`. Otherwise (and by default), the break comes before the delimiter.

To enable Auto Fill in all Fortran buffers, add `auto-fill-mode` to `fortran-mode-hook`. See Section "Hooks" in *the Emacs Manual*.

Checking Columns in Fortran

In standard Fortran 77, anything beyond column 72 is ignored. Most compilers provide an option to change this (for example, '`-ffixed-line-length-N`' in gfortran). Customize the variable `fortran-line-length` to change the line length in Fortran mode. Anything beyond this point is font-locked as a comment. (Unless it is inside a string: strings that extend beyond `fortran-line-length` will confuse font-lock.)

C-c C-r
: Display a column ruler momentarily above the current line (`fortran-column-ruler`).

C-c C-w
: Split the current window horizontally temporarily so that it is `fortran-line-length` columns wide (`fortran-window-create-momentarily`). This may help you avoid making lines longer than the limit imposed by your Fortran compiler.

C-u C-c C-w
: Split the current window horizontally so that it is `fortran-line-length` columns wide (`fortran-window-create`). You can then continue editing.

M-x fortran-strip-sequence-nos
: Delete all text in column `fortran-line-length` and beyond.

The command *C-c C-r* (`fortran-column-ruler`) shows a column ruler momentarily above the current line. The comment ruler is two lines of text that show you the locations of columns with special significance in Fortran programs. Square brackets show the limits of the columns for line numbers, and curly brackets show the limits of the columns for the statement body. Column numbers appear above them.

Note that the column numbers count from zero, as always in GNU Emacs (but customizing `column-number-indicator-zero-based` can change column display to match that of Fortran; see Section "Optional Mode Line" in *the Emacs Manual*.) As a result, the numbers may be one less than those you are familiar with; but the positions they indicate in the line are standard for Fortran.

The text used to display the column ruler depends on the value of the variable `indent-tabs-mode`. If `indent-tabs-mode` is `nil`, then the value of the variable `fortran-column-ruler-fixed` is used as the column ruler. Otherwise, the value of the variable `fortran-column-ruler-tab` is displayed. By changing these variables, you can change the column ruler display.

C-c C-w (`fortran-window-create-momentarily`) temporarily splits the current window horizontally, making a window `fortran-line-length` columns wide, so you can see any lines that are too long. Type a space to restore the normal width.

You can also split the window horizontally and continue editing with the split in place. To do this, use *C-u C-c C-w* (`M-x fortran-window-create`). By editing in this window you can immediately see when you make a line too wide to be correct Fortran.

The command *M-x fortran-strip-sequence-nos* deletes all text in column `fortran-line-length` and beyond, on all lines in the current buffer. This is the easiest way to get rid of old sequence numbers.

Fortran Keyword Abbrevs

Fortran mode provides many built-in abbrevs for common keywords and declarations. These are the same sort of abbrev that you can define yourself. To use them, you must turn on Abbrev mode. See Section "Abbrevs" in *the Emacs Manual*.

The built-in abbrevs are unusual in one way: they all start with a semicolon. For example, one built-in Fortran abbrev is '`;c`' for '`continue`'. If you insert '`;c`' and then insert a punctuation character such as a space or a newline, the '`;c`' expands automatically to '`continue`', provided Abbrev mode is enabled.

Type '`;?`' or '`;C-h`' to display a list of all the built-in Fortran abbrevs and what they stand for.

Emacs and MS-DOS

This section briefly describes the peculiarities of using Emacs on MS-DOS. Information about Emacs and Microsoft's current operating system Windows is in the main Emacs manual (see Section "Microsoft Windows" in *the Emacs Manual*).

If you build Emacs for MS-DOS, the binary will also run on Windows 3.X, Windows NT, Windows 9X/ME, or Windows 2000/XP as a DOS application; all of this chapter applies for all of those systems, if you use an Emacs that was built for MS-DOS.

See Section "Text and Binary" in *the Emacs Manual*, for information about Emacs's special handling of text files under MS-DOS (and Windows).

Keyboard Usage on MS-DOS

The key that is called `DEL` in Emacs (because that's how it is designated on most workstations) is known as `BS` (backspace) on a PC. That is why the PC-specific terminal initialization remaps the `BS` key to act as `DEL`; the `Delete` key is remapped to act as *`C-d`* for the same reasons.

Emacs built for MS-DOS recognizes *`C-Break`* as a quit character, just like *`C-g`*. This is because Emacs cannot detect that you have typed *`C-g`* until it is ready for more input. As a consequence, you cannot use *`C-g`* to stop a running command (see Section "Quitting" in *the Emacs Manual*). By contrast, *`C-Break`* *is* detected as soon as you type it (as *`C-g`* is on other systems), so it can be used to stop a running command and for emergency escape (see Section "Emergency Escape" in *the Emacs Manual*).

The PC keyboard maps use the left `Alt` key as the `Meta` key. You have two choices for emulating the `SUPER` and `Hyper` keys: choose either the right `Ctrl` key or the right `Alt` key by setting the variables `dos-hyper-key` and `dos-super-key` to 1 or 2 respectively. If neither `dos-super-key` nor `dos-hyper-key` is 1, then by default the right `Alt` key is also mapped to the `Meta` key. However, if the MS-DOS international keyboard support program `KEYB.COM` is installed, Emacs will *not* map the right `Alt` to `Meta`, since it is used for accessing characters like `~` and `@` on non-US keyboard layouts; in this case, you may only use the left `Alt` as `Meta` key.

The variable `dos-keypad-mode` is a flag variable that controls what key codes are returned by keys in the numeric keypad. You can also define the keypad `ENTER` key to act like *`C-j`*, by putting the following line into your `_emacs` file:

```
;; Make the ENTER key from the numeric keypad act as C-j.
(keymap-set function-key-map "<kp-enter>" "C-j")
```

Mouse Usage on MS-DOS

Emacs on MS-DOS supports a mouse (on the default terminal only). The mouse commands work as documented, including those that use menus and the menu bar (see Section "Menu Bar" in *the Emacs Manual*). Scroll bars don't work in MS-DOS Emacs. PC mice usually have only two buttons; these act as *`mouse-1`* and *`mouse-2`*, but if you press both of them together, that has the effect of *`mouse-3`*. If the mouse does have 3 buttons, Emacs detects that at startup, and all the 3 buttons function normally, as on X.

Help strings for menu-bar and pop-up menus are displayed in the echo area when the mouse pointer moves across the menu items. Highlighting of mouse-sensitive text (see Section "Mouse References" in *the Emacs Manual*) is also supported.

Some versions of mouse drivers don't report the number of mouse buttons correctly. For example, mice with a wheel report that they have 3 buttons, but only 2 of them are passed to Emacs; the clicks on the wheel, which serves as the middle button, are not passed. In these cases, you can use the *M-x msdos-set-mouse-buttons* command to tell Emacs how many mouse buttons to expect. You could make such a setting permanent by adding this fragment to your `_emacs` init file:

```
;; Treat the mouse like a 2-button mouse.
(msdos-set-mouse-buttons 2)
```

Emacs built for MS-DOS supports clipboard operations when it runs on Windows. Commands that put text on the kill ring, or yank text from the ring, check the Windows clipboard first, just as Emacs does on the X Window System (see Section "Mouse Commands" in *the Emacs Manual*). Only the primary selection and the cut buffer are supported by MS-DOS Emacs on Windows; the secondary selection always appears as empty.

Due to the way clipboard access is implemented by Windows, the length of text you can put into the clipboard is limited by the amount of free DOS memory that is available to Emacs. Usually, up to 620KB of text can be put into the clipboard, but this limit depends on the system configuration and is lower if you run Emacs as a subprocess of another program. If the killed text does not fit, Emacs outputs a message saying so, and does not put the text into the clipboard.

Null characters also cannot be put into the Windows clipboard. If the killed text includes null characters, Emacs does not put such text into the clipboard, and displays in the echo area a message to that effect.

The variable `dos-display-scancodes`, when non-`nil`, directs Emacs to display the ASCII value and the keyboard scan code of each keystroke; this feature serves as a complement to the `view-lossage` command, for debugging.

Display on MS-DOS

Display on MS-DOS cannot use font variants, like bold or italic, but it does support multiple faces, each of which can specify a foreground and a background color. Therefore, you can get the full functionality of Emacs packages that use fonts (such as `font-lock`, Enriched Text mode, and others) by defining the relevant faces to use different colors. Use the `list-colors-display` command (see Section "Colors" in *the Emacs Manual*) and the `list-faces-display` command (see Section "Faces" in *the Emacs Manual*) to see what colors and faces are available and what they look like.

See [MS-DOS and MULE], page 42, later in this chapter, for information on how Emacs displays glyphs and characters that aren't supported by the native font built into the DOS display.

When Emacs starts, it changes the cursor shape to a solid box. This is for compatibility with other systems, where the box cursor is the default in Emacs. This default shape can be changed to a bar by specifying the `cursor-type` parameter in the variable `default-frame-alist` (see Section "Creating Frames" in *the Emacs Manual*). The MS-DOS terminal doesn't support a vertical-bar cursor, so the bar cursor is horizontal, and the *width* parameter, if specified by the frame parameters, actually determines its height. For this reason, the `bar` and `hbar` cursor types produce the same effect on MS-DOS. As an extension, the bar cursor specification can include the starting scan line of the cursor as well as its width, like this:

```
'(cursor-type bar width . start)
```

In addition, if the *width* parameter is negative, the cursor bar begins at the top of the character cell.

The MS-DOS terminal can only display a single frame at a time. The Emacs frame facilities work on MS-DOS much as they do on text terminals (see Section "Frames" in *the Emacs Manual*). When you run Emacs from a DOS window on MS-Windows, you can make the visible frame smaller than the full screen, but Emacs still cannot display more than a single frame at a time.

The `dos-mode4350` command switches the display to 43 or 50 lines, depending on your hardware; the `dos-mode25` command switches to the default 80x25 screen size.

By default, Emacs only knows how to set screen sizes of 80 columns by 25, 28, 35, 40, 43 or 50 rows. However, if your video adapter has special video modes that will switch the display to other sizes, you can have Emacs support those too. When you ask Emacs to switch the frame to *n* rows by *m* columns dimensions, it checks if there is a variable called `screen-dimensions-nxm`, and if so, uses its value (which must be an integer) as the video mode to switch to. (Emacs switches to that video mode by calling the BIOS `Set Video Mode` function with the value of `screen-dimensions-nxm` in the `AL` register.) For example, suppose your adapter will switch to 66x80 dimensions when put into video mode 85. Then you can make Emacs support this screen size by putting the following into your `_emacs` file:

```
(setq screen-dimensions-66x80 85)
```

Since Emacs on MS-DOS can only set the frame size to specific supported dimensions, it cannot honor every possible frame resizing request. When an unsupported size is requested, Emacs chooses the next larger supported size beyond the specified size. For example, if you ask for 36x80 frame, you will get 40x80 instead.

The variables `screen-dimensions-nxm` are used only when they exactly match the specified size; the search for the next larger supported size ignores them. In the above example, even if your VGA supports 38x80 dimensions and you define a variable `screen-dimensions-38x80` with a suitable value, you will still get 40x80 screen when you ask for a 36x80 frame. If you want to get the 38x80 size in this case, you can do it by setting the variable named `screen-dimensions-36x80` with the same video mode value as `screen-dimensions-38x80`.

Changing frame dimensions on MS-DOS has the effect of changing all the other frames to the new dimensions.

File Names on MS-DOS

On MS-DOS, file names are case-insensitive and limited to eight characters, plus optionally a period and three more characters. Emacs knows enough about these limitations to handle file names that were meant for other operating systems. For instance, leading dots '`.`' in file names are invalid in MS-DOS, so Emacs transparently converts them to underscores '`_`'; thus your default init file (see Section "Init File" in *the Emacs Manual*) is called `_emacs` on MS-DOS. Excess characters before or after the period are generally ignored by MS-DOS itself; thus, if you visit the file `LongFileName.EvenLongerExtension`, you will silently get `longfile.eve`, but Emacs will still display the long file name on the mode line. Other than that, it's up to you to specify file names which are valid under MS-DOS; the transparent conversion as described above only works on file names built into Emacs.

The above restrictions on the file names on MS-DOS make it almost impossible to construct the name of a backup file (see Section "Backup Names" in *the Emacs Manual*) without losing some of the original file name characters. For example, the name of a backup file for `docs.txt` is `docs.tx~` even if single backup is used.

If you run Emacs as a DOS application under Windows 9X, Windows ME, or Windows 2000/XP, you can turn on support for long file names. If you do that, Emacs doesn't truncate file names or convert them to lower case; instead, it uses the file names that you specify, verbatim. To enable long file name support, set the environment variable `LFN` to '`y`' before starting Emacs. Unfortunately, Windows NT doesn't allow DOS programs to access long file names, so Emacs built for MS-DOS will only see their short 8+3 aliases.

MS-DOS has no notion of home directory, so Emacs on MS-DOS pretends that the directory where it is installed is the value of the `HOME` environment variable. That is, if your Emacs binary, `emacs.exe`, is in the directory `c:/utils/emacs/bin`, then Emacs acts as if `HOME` were set to '`c:/utils/emacs`'. In particular, that is where Emacs looks for the init file `_emacs`. With this in mind, you can use '`~`' in file names as an alias for the home directory, as you would on GNU or Unix. You can also set `HOME` variable in the environment before starting Emacs; its value will then override the above default behavior.

Emacs on MS-DOS handles the name `/dev` specially, because of a feature in the emulator libraries of DJGPP that pretends I/O devices have names in that directory. We recommend that you avoid using an actual directory named `/dev` on any disk.

Printing and MS-DOS

Printing commands, such as `lpr-buffer` (see Section "Printing" in *the Emacs Manual*) and `ps-print-buffer` (see Section "PostScript" in *the Emacs Manual*) can work on MS-DOS by sending the output to one of the printer ports, if a POSIX-style `lpr` program is unavailable. The same Emacs variables control printing on all systems, but in some cases they have different default values on MS-DOS.

See Section "Windows Printing" in *the Emacs Manual*, for details about setting up printing to a networked printer.

Some printers expect DOS codepage encoding of non-ASCII text, even though they are connected to a Windows machine that uses a different encoding for the same locale. For example, in the Latin-1 locale, DOS uses codepage 850 whereas Windows uses codepage 1252. See [MS-DOS and MULE], page 42. When you print to such printers from Windows, you can use the *`C-x`* `RET` *`c`* (`universal-coding-system-argument`) command before *`M-x lpr-buffer`*; Emacs will then convert the text to the DOS codepage that you specify. For example, *`C-x`* `RET` *`c cp850-dos`* `RET` *`M-x lpr-region`* `RET` will print the region while converting it to the codepage 850 encoding.

For backwards compatibility, the value of `dos-printer` (`dos-ps-printer`), if it has a value, overrides the value of `printer-name` (`ps-printer-name`), on MS-DOS.

International Support on MS-DOS

Emacs on MS-DOS supports the same international character sets as it does on GNU, Unix and other platforms (see Section "International" in *the Emacs Manual*), including coding systems for converting between the different character sets. However, due to incompatibilities between MS-DOS/MS-Windows and other systems, there are several DOS-specific aspects of this support that you should be aware of. This section describes these aspects.

The description below is largely specific to the MS-DOS port of Emacs, especially where it talks about practical implications for Emacs users.

`M-x dos-codepage-setup`
: Set up Emacs display and coding systems as appropriate for the current DOS codepage.

MS-DOS is designed to support one character set of 256 characters at any given time, but gives you a variety of character sets to choose from. The alternative character sets are known as *DOS codepages*. Each codepage includes all 128 ASCII characters, but the other 128 characters (codes 128 through 255) vary from one codepage to another. Each DOS codepage is identified by a 3-digit number, such as 850, 862, etc.

In contrast to X, which lets you use several fonts at the same time, MS-DOS normally doesn't allow use of several codepages in a single session. MS-DOS was designed to load a single codepage at system startup, and require you to reboot in order to change it[1]. Much the same limitation applies when you run DOS executables on other systems such as MS-Windows.

For multibyte operation on MS-DOS, Emacs needs to know which characters the chosen DOS codepage can display. So it queries the system shortly after startup to get the chosen codepage number, and stores the number in the variable `dos-codepage`. Some systems return the default value 437 for the current codepage, even though the actual codepage is different. (This typically happens when you use the codepage built into the display hardware.) You can specify a different codepage for Emacs to use by setting the variable `dos-codepage` in your init file.

Multibyte Emacs supports only certain DOS codepages: those which can display Far-Eastern scripts, like the Japanese codepage 932, and those that encode a single ISO 8859 character set.

The Far-Eastern codepages can directly display one of the MULE character sets for these countries, so Emacs simply sets up to use the appropriate terminal coding system that is supported by the codepage. The special features described in the rest of this section mostly pertain to codepages that encode ISO 8859 character sets.

For the codepages that correspond to one of the ISO character sets, Emacs knows the character set based on the codepage number. Emacs automatically creates a coding system to support reading and writing files that use the current codepage, and uses this coding

---

[1] Normally, one particular codepage is burnt into the display memory, while other codepages can be installed by modifying system configuration files, such as `CONFIG.SYS`, and rebooting. While there is third-party software that allows changing the codepage without rebooting, we describe here how a stock MS-DOS system behaves.

system by default. The name of this coding system is `cpnnn`, where *nnn* is the codepage number.[2]

All the `cpnnn` coding systems use the letter 'D' (for "DOS") as their mode-line mnemonic. Since both the terminal coding system and the default coding system for file I/O are set to the proper `cpnnn` coding system at startup, it is normal for the mode line on MS-DOS to begin with '-DD\-'. See Section "Mode Line" in *the Emacs Manual*. Far-Eastern DOS terminals do not use the `cpnnn` coding systems, and thus their initial mode line looks like the Emacs default.

Since the codepage number also indicates which script you are using, Emacs automatically runs `set-language-environment` to select the language environment for that script (see Section "Language Environments" in *the Emacs Manual*).

If a buffer contains a character belonging to some other ISO 8859 character set, not the one that the chosen DOS codepage supports, Emacs displays it using a sequence of ASCII characters. For example, if the current codepage doesn't have a glyph for the letter 'ò' (small 'o' with a grave accent), it is displayed as '{`o}', where the braces serve as a visual indication that this is a single character. (This may look awkward for some non-Latin characters, such as those from Greek or Hebrew alphabets, but it is still readable by a person who knows the language.) Even though the character may occupy several columns on the screen, it is really still just a single character, and all Emacs commands treat it as one.

MS-Windows provides its own codepages, which are different from the DOS codepages for the same locale. For example, DOS codepage 850 supports the same character set as Windows codepage 1252; DOS codepage 855 supports the same character set as Windows codepage 1251, etc. The MS-Windows version of Emacs uses the current codepage for display when invoked with the '-nw' option.

Subprocesses on MS-DOS

Because MS-DOS is a single-process "operating system", asynchronous subprocesses are not available. In particular, Shell mode and its variants do not work. Most Emacs features that use asynchronous subprocesses also don't work on MS-DOS, including Shell mode and GUD. When in doubt, try and see; commands that don't work output an error message saying that asynchronous processes aren't supported.

Compilation under Emacs with *M-x compile*, searching files with *M-x grep* and displaying differences between files with *M-x diff* do work, by running the inferior processes synchronously. This means you cannot do any more editing until the inferior process finishes.

Spell checking also works, by means of special support for synchronous invocation of the `ispell` program. This is slower than the asynchronous invocation on other platforms.

Instead of the Shell mode, which doesn't work on MS-DOS, you can use the *M-x eshell* command. This invokes the Eshell package that implements a POSIX-like shell entirely in Emacs Lisp.

[2] The standard Emacs coding systems for ISO 8859 are not quite right for the purpose, because typically the DOS codepage does not match the standard ISO character codes. For example, the letter 'ç' ('c' with cedilla) has code 231 in the standard Latin-1 character set, but the corresponding DOS codepage 850 uses code 135 for this glyph.

By contrast, Emacs compiled as a native Windows application **does** support asynchronous subprocesses. See Section “Windows Processes” in *the Emacs Manual*.

Printing commands, such as `lpr-buffer` (see Section “Printing” in *the Emacs Manual*) and `ps-print-buffer` (see Section “PostScript” in *the Emacs Manual*), work in MS-DOS by sending the output to one of the printer ports. See Section “MS-DOS Printing” in *the Emacs Manual*.

When you run a subprocess synchronously on MS-DOS, make sure the program terminates and does not try to read keyboard input. If the program does not terminate on its own, you will be unable to terminate it, because MS-DOS provides no general way to terminate a process. Pressing *C-c* or *C-*`Break` might sometimes help in these cases.

Accessing files on other machines is not supported on MS-DOS. Other network-oriented commands such as sending mail, Web browsing, remote login, etc., don’t work either, unless network access is built into MS-DOS with some network redirector.

Dired on MS-DOS uses the `ls-lisp` package (see Section “ls in Lisp” in *the Emacs Manual*). Therefore, Dired on MS-DOS supports only some of the possible options you can mention in the `dired-listing-switches` variable. The options that work are ‘`-A`’, ‘`-a`’, ‘`-c`’, ‘`-i`’, ‘`-r`’, ‘`-S`’, ‘`-s`’, ‘`-t`’, and ‘`-u`’.

Appendix A GNU Free Documentation License

Version 1.3, 3 November 2008

0. PREAMBLE

   The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

   This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

   We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

   This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

   A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

   A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

   The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `https://www.gnu.org/licenses/`.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

    "Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

    "CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

    "Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

    An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

    The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

V

W

Y