GNU Emacs Manual

Nineteenth Edition, Updated for Emacs Version 27.2.

Richard Stallman et al.

19 International Character Set Support

Emacs supports a wide variety of international character sets, including European and Vietnamese variants of the Latin alphabet, as well as Arabic scripts, Brahmic scripts (for languages such as Bengali, Hindi, and Thai), Cyrillic, Ethiopic, Georgian, Greek, Han (for Chinese and Japanese), Hangul (for Korean), Hebrew and IPA. Emacs also supports various encodings of these characters that are used by other internationalized software, such as word processors and mailers.

Emacs allows editing text with international characters by supporting all the related activities:

- You can visit files with non-ASCII characters, save non-ASCII text, and pass non-ASCII text between Emacs and programs it invokes (such as compilers, spell-checkers, and mailers). Setting your language environment (see Section 19.2 [Language Environments], page 203) takes care of setting up the coding systems and other options for a specific language or culture. Alternatively, you can specify how Emacs should encode or decode text for each command; see Section 19.9 [Text Coding], page 212.
- You can display non-ASCII characters encoded by the various scripts. This works by using appropriate fonts on graphics displays (see Section 19.14 [Defining Fontsets], page 217), and by sending special codes to text displays (see Section 19.12 [Terminal Coding], page 215). If some characters are displayed incorrectly, refer to Section 19.16 [Undisplayable Characters], page 220, which describes possible problems and explains how to solve them.
- Characters from scripts whose natural ordering of text is from right to left are reordered for display (see Section 19.19 [Bidirectional Editing], page 222). These scripts include Arabic, Hebrew, Syriac, Thaana, and a few others.
- You can insert non-ASCII characters or search for them. To do that, you can specify an input method (see Section 19.4 [Select Input Method], page 206) suitable for your language, or use the default input method set up when you choose your language environment. If your keyboard can produce non-ASCII characters, you can select an appropriate keyboard coding system (see Section 19.12 [Terminal Coding], page 215), and Emacs will accept those characters. Latin-1 characters can also be input by using the `C-x 8` prefix, see Section 19.17 [Unibyte Mode], page 220.

  With the X Window System, your locale should be set to an appropriate value to make sure Emacs interprets keyboard input correctly; see Section 19.2 [Language Environments], page 203.

The rest of this chapter describes these issues in detail.

19.1 Introduction to International Character Sets

The users of international character sets and scripts have established many more-or-less standard coding systems for storing files. These coding systems are typically *multibyte*, meaning that sequences of two or more bytes are used to represent individual non-ASCII characters.

Internally, Emacs uses its own multibyte character encoding, which is a superset of the *Unicode* standard. This internal encoding allows characters from almost every known

script to be intermixed in a single buffer or string. Emacs translates between the multibyte character encoding and various other coding systems when reading and writing files, and when exchanging data with subprocesses.

The command `C-h h` (`view-hello-file`) displays the file `etc/HELLO`, which illustrates various scripts by showing how to say "hello" in many languages. If some characters can't be displayed on your terminal, they appear as '`?`' or as hollow boxes (see Section 19.16 [Undisplayable Characters], page 220).

Keyboards, even in the countries where these character sets are used, generally don't have keys for all the characters in them. You can insert characters that your keyboard does not support, using `C-x 8 RET` (`insert-char`). See Section 4.1 [Inserting Text], page 16. Shorthands are available for some common characters; for example, you can insert a left single quotation mark '`‘`' by typing `C-x 8 [`, or in Electric Quote mode, usually by simply typing '`` ` ``'. See Section 22.5 [Quotation Marks], page 238. Emacs also supports various *input methods*, typically one for each script or language, which make it easier to type characters in the script. See Section 19.3 [Input Methods], page 205.

The prefix key `C-x RET` is used for commands that pertain to multibyte characters, coding systems, and input methods.

The command `C-x =` (`what-cursor-position`) shows information about the character at point. In addition to the character position, which was described in Section 4.9 [Position Info], page 22, this command displays how the character is encoded. For instance, it displays the following line in the echo area for the character '`c`':

```
Char: c (99, #o143, #x63) point=28062 of 36168 (78%) column=53
```

The four values after '`Char:`' describe the character that follows point, first by showing it and then by giving its character code in decimal, octal and hex. For a non-ASCII multibyte character, these are followed by '`file`' and the character's representation, in hex, in the buffer's coding system, if that coding system encodes the character safely and with a single byte (see Section 19.5 [Coding Systems], page 208). If the character's encoding is longer than one byte, Emacs shows '`file ...`'.

On rare occasions, Emacs encounters *raw bytes*: single bytes whose values are in the range 128 (0200 octal) through 255 (0377 octal), which Emacs cannot interpret as part of a known encoding of some non-ASCII character. Such raw bytes are treated as if they belonged to a special character set `eight-bit`; Emacs displays them as escaped octal codes (this can be customized; see Section 11.23 [Display Custom], page 94). In this case, `C-x =` shows '`raw-byte`' instead of '`file`'. In addition, `C-x =` shows the character codes of raw bytes as if they were in the range `#x3FFF80..#x3FFFFF`, which is where Emacs maps them to distinguish them from Unicode characters in the range `#x0080..#x00FF`.

With a prefix argument (`C-u C-x =`), this command additionally calls the command `describe-char`, which displays a detailed description of the character:

- The character set name, and the codes that identify the character within that character set; ASCII characters are identified as belonging to the `ascii` character set.
- The character's script, syntax and categories.
- What keys to type to input the character in the current input method (if it supports the character).
- The character's encodings, both internally in the buffer, and externally if you were to save the buffer to a file.

- If you are running Emacs on a graphical display, the font name and glyph code for the character. If you are running Emacs on a text terminal, the code(s) sent to the terminal.
- If the character was composed on display with any following characters to form one or more grapheme clusters, the composition information: the font glyphs if the frame is on a graphical display, else the characters that were composed.
- The character's text properties (see Section "Text Properties" in *the Emacs Lisp Reference Manual*), including any non-default faces used to display the character, and any overlays containing it (see Section "Overlays" in *the same manual*).

Here's an example, with some lines folded to fit into this manual:

```
              position: 1 of 1 (0%), column: 0
             character: ê (displayed as ê) (codepoint 234, #o352, #xea)
    preferred charset: unicode (Unicode (ISO10646))
code point in charset: 0xEA
               script: latin
               syntax: w        which means: word
             category: .:Base, L:Left-to-right (strong), c:Chinese,
                       j:Japanese, l:Latin, v:Viet
             to input: type "C-x 8 RET ea" or
                       "C-x 8 RET LATIN SMALL LETTER E WITH CIRCUMFLEX"
          buffer code: #xC3 #xAA
            file code: #xC3 #xAA (encoded by coding system utf-8-unix)
              display: by this font (glyph code)
    xft:-PfEd-DejaVu Sans Mono-normal-normal-
        normal-*-15-*-*-*-m-0-iso10646-1 (#xAC)

Character code properties: customize what to show
  name: LATIN SMALL LETTER E WITH CIRCUMFLEX
  old-name: LATIN SMALL LETTER E CIRCUMFLEX
  general-category: Ll (Letter, Lowercase)
  decomposition: (101 770) ('e' '^')
```

19.2 Language Environments

All supported character sets are supported in Emacs buffers whenever multibyte characters are enabled; there is no need to select a particular language in order to display its characters. However, it is important to select a *language environment* in order to set various defaults. Roughly speaking, the language environment represents a choice of preferred script rather than a choice of language.

The language environment controls which coding systems to recognize when reading text (see Section 19.6 [Recognize Coding], page 209). This applies to files, incoming mail, and any other text you read into Emacs. It may also specify the default coding system to use when you create a file. Each language environment also specifies a default input method.

To select a language environment, customize `current-language-environment` or use the command `M-x set-language-environment`. It makes no difference which buffer is current when you use this command, because the effects apply globally to the Emacs session. See the variable `language-info-alist` for the list of supported language environments, and use the command `C-h L` *`lang-env`* `RET` (`describe-language-environment`) for more information about the language environment *lang-env*. Supported language environments include:

ASCII, Arabic, Belarusian, Bengali, Brazilian Portuguese, Bulgarian, Burmese, Cham, Chinese-BIG5, Chinese-CNS, Chinese-EUC-TW, Chinese-GB, Chinese-GB18030, Chinese-GBK, Croatian, Cyrillic-ALT, Cyrillic-ISO, Cyrillic-KOI8, Czech, Devanagari, Dutch, English, Esperanto, Ethiopic, French, Georgian, German, Greek, Gujarati, Hebrew, IPA, Italian, Japanese, Kannada, Khmer, Korean, Lao, Latin-1, Latin-2, Latin-3, Latin-4, Latin-5, Latin-6, Latin-7, Latin-8, Latin-9, Latvian, Lithuanian, Malayalam, Oriya, Persian, Polish, Punjabi, Romanian, Russian, Sinhala, Slovak, Slovenian, Spanish, Swedish, TaiViet, Tajik, Tamil, Telugu, Thai, Tibetan, Turkish, UTF-8, Ukrainian, Vietnamese, Welsh, and Windows-1255.

To display the script(s) used by your language environment on a graphical display, you need to have suitable fonts. See Section 19.13 [Fontsets], page 216, for more details about setting up your fonts.

Some operating systems let you specify the character-set locale you are using by setting the locale environment variables `LC_ALL`, `LC_CTYPE`, or `LANG`. (If more than one of these is set, the first one that is nonempty specifies your locale for this purpose.) During startup, Emacs looks up your character-set locale's name in the system locale alias table, matches its canonical name against entries in the value of the variables `locale-charset-language-names` and `locale-language-names` (the former overrides the latter), and selects the corresponding language environment if a match is found. It also adjusts the display table and terminal coding system, the locale coding system, the preferred coding system as needed for the locale, and—last but not least—the way Emacs decodes non-ASCII characters sent by your keyboard.

If you modify the `LC_ALL`, `LC_CTYPE`, or `LANG` environment variables while running Emacs (by using `M-x setenv`), you may want to invoke the `set-locale-environment` command afterwards to readjust the language environment from the new locale.

The `set-locale-environment` function normally uses the preferred coding system established by the language environment to decode system messages. But if your locale matches an entry in the variable `locale-preferred-coding-systems`, Emacs uses the corresponding coding system instead. For example, if the locale '`ja_JP.PCK`' matches `japanese-shift-jis` in `locale-preferred-coding-systems`, Emacs uses that encoding even though it might normally use `utf-8`.

You can override the language environment chosen at startup with explicit use of the command `set-language-environment`, or with customization of `current-language-environment` in your init file.

To display information about the effects of a certain language environment *lang-env*, use the command `C-h L` *lang-env* `RET` (`describe-language-environment`). This tells you which languages this language environment is useful for, and lists the character sets, coding systems, and input methods that go with it. It also shows some sample text to illustrate scripts used in this language environment. If you give an empty input for *lang-env*, this command describes the chosen language environment.

You can customize any language environment with the normal hook `set-language-environment-hook`. The command `set-language-environment` runs that hook after setting up the new language environment. The hook functions can test for a specific language environment by checking the variable `current-language-environment`. This hook is where

you should put non-default settings for specific language environments, such as coding systems for keyboard input and terminal output, the default input method, etc.

Before it starts to set up the new language environment, `set-language-environment` first runs the hook `exit-language-environment-hook`. This hook is useful for undoing customizations that were made with `set-language-environment-hook`. For instance, if you set up a special key binding in a specific language environment using `set-language-environment-hook`, you should set up `exit-language-environment-hook` to restore the normal binding for that key.

19.3 Input Methods

An *input method* is a kind of character conversion designed specifically for interactive input. In Emacs, typically each language has its own input method; sometimes several languages that use the same characters can share one input method. A few languages support several input methods.

The simplest kind of input method works by mapping ASCII letters into another alphabet; this allows you to use one other alphabet instead of ASCII. The Greek and Russian input methods work this way.

A more powerful technique is composition: converting sequences of characters into one letter. Many European input methods use composition to produce a single non-ASCII letter from a sequence that consists of a letter followed by accent characters (or vice versa). For example, some methods convert the sequence `o ^` into a single accented letter. These input methods have no special commands of their own; all they do is compose sequences of printing characters.

The input methods for syllabic scripts typically use mapping followed by composition. The input methods for Thai and Korean work this way. First, letters are mapped into symbols for particular sounds or tone marks; then, sequences of these that make up a whole syllable are mapped into one syllable sign.

Chinese and Japanese require more complex methods. In Chinese input methods, first you enter the phonetic spelling of a Chinese word (in input method `chinese-py`, among others), or a sequence of portions of the character (input methods `chinese-4corner` and `chinese-sw`, and others). One input sequence typically corresponds to many possible Chinese characters. You select the one you mean using keys such as `C-f`, `C-b`, `C-n`, `C-p` (or the arrow keys), and digits, which have special meanings in this situation.

The possible characters are conceptually arranged in several rows, with each row holding up to 10 alternatives. Normally, Emacs displays just one row at a time, in the echo area; `(`*i*`/`*j*`)` appears at the beginning, to indicate that this is the *i*th row out of a total of *j* rows. Type `C-n` or `C-p` to display the next row or the previous row.

Type `C-f` and `C-b` to move forward and backward among the alternatives in the current row. As you do this, Emacs highlights the current alternative with a special color; type `C-SPC` to select the current alternative and use it as input. The alternatives in the row are also numbered; the number appears before the alternative. Typing a number selects the associated alternative of the current row and uses it as input.

`TAB` in these Chinese input methods displays a buffer showing all the possible characters at once; then clicking `mouse-2` on one of them selects that alternative. The keys `C-f`, `C-b`,

`C-n`, `C-p`, and digits continue to work as usual, but they do the highlighting in the buffer showing the possible characters, rather than in the echo area.

To enter characters according to the *pīnyīn* transliteration method instead, use the `chinese-sisheng` input method. This is a composition based method, where e.g. `pi1` results in '`pī`'.

In Japanese input methods, first you input a whole word using phonetic spelling; then, after the word is in the buffer, Emacs converts it into one or more characters using a large dictionary. One phonetic spelling corresponds to a number of different Japanese words; to select one of them, use `C-n` and `C-p` to cycle through the alternatives.

Sometimes it is useful to cut off input method processing so that the characters you have just entered will not combine with subsequent characters. For example, in input method `latin-1-postfix`, the sequence `o ^` combines to form an '`o`' with an accent. What if you want to enter them as separate characters?

One way is to type the accent twice; this is a special feature for entering the separate letter and accent. For example, `o ^ ^` gives you the two characters '`o^`'. Another way is to type another letter after the `o`—something that won't combine with that—and immediately delete it. For example, you could type `o o DEL ^` to get separate '`o`' and '`^`'. Another method, more general but not quite as easy to type, is to use `C-\ C-\` between two characters to stop them from combining. This is the command `C-\` (`toggle-input-method`) used twice.

`C-\ C-\` is especially useful inside an incremental search, because it stops waiting for more characters to combine, and starts searching for what you have already entered.

To find out how to input the character after point using the current input method, type `C-u C-x =`. See Section 4.9 [Position Info], page 22.

The variables `input-method-highlight-flag` and `input-method-verbose-flag` control how input methods explain what is happening. If `input-method-highlight-flag` is non-`nil`, the partial sequence is highlighted in the buffer (for most input methods—some disable this feature). If `input-method-verbose-flag` is non-`nil`, the list of possible characters to type next is displayed in the echo area (but not when you are in the minibuffer).

You can modify how an input method works by making your changes in a function that you add to the hook variable `quail-activate-hook`. See Section 33.2.2 [Hooks], page 469. For example, you can redefine some of the input method's keys by defining key bindings in the keymap returned by the function `quail-translation-keymap`, using `define-key`. See Section 33.3.6 [Init Rebinding], page 480.

Input methods are inhibited when the text in the buffer is read-only for some reason. This is so single-character key bindings work in modes that make buffer text or parts of it read-only, such as `read-only-mode` and `image-mode`, even when an input method is active.

Another facility for typing characters not on your keyboard is by using `C-x 8 RET` (`insert-char`) to insert a single character based on its Unicode name or code-point; see Section 4.1 [Inserting Text], page 16.

19.4 Selecting an Input Method

`C-\` Enable or disable use of the selected input method (`toggle-input-method`).

`C-x RET C-\` *method* `RET`
Select a new input method for the current buffer (`set-input-method`).

`C-h I` *method* `RET`
`C-h C-\` *method* `RET`
: Describe the input method *method* (`describe-input-method`). By default, it describes the current input method (if any). This description should give you the full details of how to use any particular input method.

`M-x list-input-methods`
: Display a list of all the supported input methods.

To choose an input method for the current buffer, use `C-x RET C-\` (`set-input-method`). This command reads the input method name from the minibuffer; the name normally starts with the language environment that it is meant to be used with. The variable `current-input-method` records which input method is selected.

Input methods use various sequences of ASCII characters to stand for non-ASCII characters. Sometimes it is useful to turn off the input method temporarily. To do this, type `C-\` (`toggle-input-method`). To reenable the input method, type `C-\` again.

If you type `C-\` and you have not yet selected an input method, it prompts you to specify one. This has the same effect as using `C-x RET C-\` to specify an input method.

When invoked with a numeric argument, as in `C-u C-\`, `toggle-input-method` always prompts you for an input method, suggesting the most recently selected one as the default.

Selecting a language environment specifies a default input method for use in various buffers. When you have a default input method, you can select it in the current buffer by typing `C-\`. The variable `default-input-method` specifies the default input method (`nil` means there is none).

In some language environments, which support several different input methods, you might want to use an input method different from the default chosen by `set-language-environment`. You can instruct Emacs to select a different default input method for a certain language environment, if you wish, by using `set-language-environment-hook` (see Section 19.2 [Language Environments], page 203). For example:

```
(defun my-chinese-setup ()
  "Set up my private Chinese environment."
  (if (equal current-language-environment "Chinese-GB")
      (setq default-input-method "chinese-tonepy")))
(add-hook 'set-language-environment-hook 'my-chinese-setup)
```

This sets the default input method to be `chinese-tonepy` whenever you choose a Chinese-GB language environment.

You can instruct Emacs to activate a certain input method automatically. For example:

```
(add-hook 'text-mode-hook
  (lambda () (set-input-method "german-prefix")))
```

This automatically activates the input method `german-prefix` in Text mode.

Some input methods for alphabetic scripts work by (in effect) remapping the keyboard to emulate various keyboard layouts commonly used for those scripts. How to do this remapping properly depends on your actual keyboard layout. To specify which layout your keyboard has, use the command `M-x quail-set-keyboard-layout`.

You can use the command `M-x quail-show-key` to show what key (or key sequence) to type in order to input the character following point, using the selected keyboard layout. The

command `C-u C-x =` also shows that information, in addition to other information about the character.

`M-x list-input-methods` displays a list of all the supported input methods. The list gives information about each input method, including the string that stands for it in the mode line.

19.5 Coding Systems

Users of various languages have established many more-or-less standard coding systems for representing them. Emacs does not use these coding systems internally; instead, it converts from various coding systems to its own system when reading data, and converts the internal coding system to other coding systems when writing data. Conversion is possible in reading or writing files, in sending or receiving from the terminal, and in exchanging data with subprocesses.

Emacs assigns a name to each coding system. Most coding systems are used for one language, and the name of the coding system starts with the language name. Some coding systems are used for several languages; their names usually start with '`iso`'. There are also special coding systems, such as `no-conversion`, `raw-text`, and `emacs-internal`.

A special class of coding systems, collectively known as *codepages*, is designed to support text encoded by MS-Windows and MS-DOS software. The names of these coding systems are `cp`*nnnn*, where *nnnn* is a 3- or 4-digit number of the codepage. You can use these encodings just like any other coding system; for example, to visit a file encoded in codepage 850, type `C-x RET c cp850 RET C-x C-f` *filename* `RET`.

In addition to converting various representations of non-ASCII characters, a coding system can perform end-of-line conversion. Emacs handles three different conventions for how to separate lines in a file: newline (Unix), carriage return followed by linefeed (DOS), and just carriage return (Mac).

`C-h C` *coding* `RET`
: Describe coding system *coding* (`describe-coding-system`).

`C-h C RET` Describe the coding systems currently in use (`describe-coding-system`).

`M-x list-coding-systems`
: Display a list of all the supported coding systems.

The command `C-h C` (`describe-coding-system`) displays information about particular coding systems, including the end-of-line conversion specified by those coding systems. You can specify a coding system name as the argument; alternatively, with an empty argument, it describes the coding systems currently selected for various purposes, both in the current buffer and as the defaults, and the priority list for recognizing coding systems (see Section 19.6 [Recognize Coding], page 209).

To display a list of all the supported coding systems, type `M-x list-coding-systems`. The list gives information about each coding system, including the letter that stands for it in the mode line (see Section 1.3 [Mode Line], page 8).

Each of the coding systems that appear in this list—except for `no-conversion`, which means no conversion of any kind—specifies how and whether to convert printing characters, but leaves the choice of end-of-line conversion to be decided based on the contents of each

file. For example, if the file appears to use the sequence carriage return and linefeed to separate lines, DOS end-of-line conversion will be used.

Each of the listed coding systems has three variants, which specify exactly what to do for end-of-line conversion:

`...-unix` Don't do any end-of-line conversion; assume the file uses newline to separate lines. (This is the convention normally used on Unix and GNU systems, and macOS.)

`...-dos` Assume the file uses carriage return followed by linefeed to separate lines, and do the appropriate conversion. (This is the convention normally used on Microsoft systems.[1])

`...-mac` Assume the file uses carriage return to separate lines, and do the appropriate conversion. (This was the convention used in Classic Mac OS.)

These variant coding systems are omitted from the `list-coding-systems` display for brevity, since they are entirely predictable. For example, the coding system `iso-latin-1` has variants `iso-latin-1-unix`, `iso-latin-1-dos` and `iso-latin-1-mac`.

The coding systems `unix`, `dos`, and `mac` are aliases for `undecided-unix`, `undecided-dos`, and `undecided-mac`, respectively. These coding systems specify only the end-of-line conversion, and leave the character code conversion to be deduced from the text itself.

The coding system `raw-text` is good for a file which is mainly ASCII text, but may contain byte values above 127 that are not meant to encode non-ASCII characters. With `raw-text`, Emacs copies those byte values unchanged, and sets `enable-multibyte-characters` to `nil` in the current buffer so that they will be interpreted properly. `raw-text` handles end-of-line conversion in the usual way, based on the data encountered, and has the usual three variants to specify the kind of end-of-line conversion to use.

In contrast, the coding system `no-conversion` specifies no character code conversion at all—none for non-ASCII byte values and none for end of line. This is useful for reading or writing binary files, tar files, and other files that must be examined verbatim. It, too, sets `enable-multibyte-characters` to `nil`.

The easiest way to edit a file with no conversion of any kind is with the `M-x find-file-literally` command. This uses `no-conversion`, and also suppresses other Emacs features that might convert the file contents before you see them. See Section 15.2 [Visiting], page 136.

The coding system `emacs-internal` (or `utf-8-emacs`, which is equivalent) means that the file contains non-ASCII characters stored with the internal Emacs encoding. This coding system handles end-of-line conversion based on the data encountered, and has the usual three variants to specify the kind of end-of-line conversion.

19.6 Recognizing Coding Systems

Whenever Emacs reads a given piece of text, it tries to recognize which coding system to use. This applies to files being read, output from subprocesses, text from X selections, etc.

[1] It is also specified for MIME '`text/*`' bodies and in other network transport contexts. It is different from the SGML reference syntax record-start/record-end format, which Emacs doesn't support directly.

Emacs can select the right coding system automatically most of the time—once you have specified your preferences.

Some coding systems can be recognized or distinguished by which byte sequences appear in the data. However, there are coding systems that cannot be distinguished, not even potentially. For example, there is no way to distinguish between Latin-1 and Latin-2; they use the same byte values with different meanings.

Emacs handles this situation by means of a priority list of coding systems. Whenever Emacs reads a file, if you do not specify the coding system to use, Emacs checks the data against each coding system, starting with the first in priority and working down the list, until it finds a coding system that fits the data. Then it converts the file contents assuming that they are represented in this coding system.

The priority list of coding systems depends on the selected language environment (see Section 19.2 [Language Environments], page 203). For example, if you use French, you probably want Emacs to prefer Latin-1 to Latin-2; if you use Czech, you probably want Latin-2 to be preferred. This is one of the reasons to specify a language environment.

However, you can alter the coding system priority list in detail with the command `M-x prefer-coding-system`. This command reads the name of a coding system from the minibuffer, and adds it to the front of the priority list, so that it is preferred to all others. If you use this command several times, each use adds one element to the front of the priority list.

If you use a coding system that specifies the end-of-line conversion type, such as `iso-8859-1-dos`, what this means is that Emacs should attempt to recognize `iso-8859-1` with priority, and should use DOS end-of-line conversion when it does recognize `iso-8859-1`.

Sometimes a file name indicates which coding system to use for the file. The variable `file-coding-system-alist` specifies this correspondence. There is a special function `modify-coding-system-alist` for adding elements to this list. For example, to read and write all '`.txt`' files using the coding system `chinese-iso-8bit`, you can execute this Lisp expression:

```
(modify-coding-system-alist 'file "\\.txt\\'" 'chinese-iso-8bit)
```

The first argument should be `file`, the second argument should be a regular expression that determines which files this applies to, and the third argument says which coding system to use for these files.

Emacs recognizes which kind of end-of-line conversion to use based on the contents of the file: if it sees only carriage returns, or only carriage return followed by linefeed sequences, then it chooses the end-of-line conversion accordingly. You can inhibit the automatic use of end-of-line conversion by setting the variable `inhibit-eol-conversion` to non-`nil`. If you do that, DOS-style files will be displayed with the '`^M`' characters visible in the buffer; some people prefer this to the more subtle '`(DOS)`' end-of-line type indication near the left edge of the mode line (see Section 1.3 [Mode Line], page 8).

By default, the automatic detection of the coding system is sensitive to escape sequences. If Emacs sees a sequence of characters that begin with an escape character, and the sequence is valid as an ISO-2022 code, that tells Emacs to use one of the ISO-2022 encodings to decode the file.

However, there may be cases that you want to read escape sequences in a file as is. In such a case, you can set the variable `inhibit-iso-escape-detection` to non-`nil`. Then the code detection ignores any escape sequences, and never uses an ISO-2022 encoding. The result is that all escape sequences become visible in the buffer.

The default value of `inhibit-iso-escape-detection` is `nil`. We recommend that you not change it permanently, only for one specific operation. That's because some Emacs Lisp source files in the Emacs distribution contain non-ASCII characters encoded in the coding system `iso-2022-7bit`, and they won't be decoded correctly when you visit those files if you suppress the escape sequence detection.

The variables `auto-coding-alist` and `auto-coding-regexp-alist` are the strongest way to specify the coding system for certain patterns of file names, or for files containing certain patterns, respectively. These variables even override '`-*-coding:-*-`' tags in the file itself (see Section 19.7 [Specify Coding], page 211). For example, Emacs uses `auto-coding-alist` for tar and archive files, to prevent it from being confused by a '`-*-coding:-*-`' tag in a member of the archive and thinking it applies to the archive file as a whole.

Another way to specify a coding system is with the variable `auto-coding-functions`. For example, one of the builtin `auto-coding-functions` detects the encoding for XML files. Unlike the previous two, this variable does not override any '`-*-coding:-*-`' tag.

19.7 Specifying a File's Coding System

If Emacs recognizes the encoding of a file incorrectly, you can reread the file using the correct coding system with `C-x RET r` (`revert-buffer-with-coding-system`). This command prompts for the coding system to use. To see what coding system Emacs actually used to decode the file, look at the coding system mnemonic letter near the left edge of the mode line (see Section 1.3 [Mode Line], page 8), or type `C-h C` (`describe-coding-system`).

You can specify the coding system for a particular file in the file itself, using the '`-*-...-*-`' construct at the beginning, or a local variables list at the end (see Section 33.2.4 [File Variables], page 472). You do this by defining a value for the "variable" named `coding`. Emacs does not really have a variable `coding`; instead of setting a variable, this uses the specified coding system for the file. For example, '`-*-mode: C; coding: latin-1; -*-`' specifies use of the Latin-1 coding system, as well as C mode. When you specify the coding explicitly in the file, that overrides `file-coding-system-alist`.

19.8 Choosing Coding Systems for Output

Once Emacs has chosen a coding system for a buffer, it stores that coding system in `buffer-file-coding-system`. That makes it the default for operations that write from this buffer into a file, such as `save-buffer` and `write-region`. You can specify a different coding system for further file output from the buffer using `set-buffer-file-coding-system` (see Section 19.9 [Text Coding], page 212).

You can insert any character Emacs supports into any Emacs buffer, but most coding systems can only handle a subset of these characters. Therefore, it's possible that the characters you insert cannot be encoded with the coding system that will be used to save the buffer. For example, you could visit a text file in Polish, encoded in `iso-8859-2`, and add some Russian words to it. When you save that buffer, Emacs cannot use the

current value of `buffer-file-coding-system`, because the characters you added cannot be encoded by that coding system.

When that happens, Emacs tries the most-preferred coding system (set by `M-x prefer-coding-system` or `M-x set-language-environment`). If that coding system can safely encode all of the characters in the buffer, Emacs uses it, and stores its value in `buffer-file-coding-system`. Otherwise, Emacs displays a list of coding systems suitable for encoding the buffer's contents, and asks you to choose one of those coding systems.

If you insert the unsuitable characters in a mail message, Emacs behaves a bit differently. It additionally checks whether the most-preferred coding system is recommended for use in MIME messages; if not, it informs you of this fact and prompts you for another coding system. This is so you won't inadvertently send a message encoded in a way that your recipient's mail software will have difficulty decoding. (You can still use an unsuitable coding system if you enter its name at the prompt.)

When you send a mail message (see Chapter 29 [Sending Mail], page 388), Emacs has four different ways to determine the coding system to use for encoding the message text. It first tries the buffer's own value of `buffer-file-coding-system`, if that is non-`nil`. Otherwise, it uses the value of `sendmail-coding-system`, if that is non-`nil`. Thirdly, it uses the value of `default-sendmail-coding-system`. If all of these three values are `nil`, Emacs encodes outgoing mail using the default coding system for new files (i.e., the default value of `buffer-file-coding-system`), which is controlled by your choice of language environment.

19.9 Specifying a Coding System for File Text

In cases where Emacs does not automatically choose the right coding system for a file's contents, you can use these commands to specify one:

`C-x RET f` *coding* `RET`
: Use coding system *coding* to save or revisit the file in the current buffer (`set-buffer-file-coding-system`).

`C-x RET c` *coding* `RET`
: Specify coding system *coding* for the immediately following command (`universal-coding-system-argument`).

`C-x RET r` *coding* `RET`
: Revisit the current file using the coding system *coding* (`revert-buffer-with-coding-system`).

`M-x recode-region RET` *right* `RET` *wrong* `RET`
: Convert a region that was decoded using coding system *wrong*, decoding it using coding system *right* instead.

The command `C-x RET f` (`set-buffer-file-coding-system`) sets the file coding system for the current buffer (i.e., the coding system to use when saving or reverting the file). You specify which coding system using the minibuffer. You can also invoke this command by clicking with `mouse-3` on the coding system indicator in the mode line (see Section 1.3 [Mode Line], page 8).

If you specify a coding system that cannot handle all the characters in the buffer, Emacs will warn you about the troublesome characters, and ask you to choose another coding system, when you try to save the buffer (see Section 19.8 [Output Coding], page 211).

You can also use this command to specify the end-of-line conversion (see Section 19.5 [Coding Systems], page 208) for encoding the current buffer. For example, `C-x RET f dos RET` will cause Emacs to save the current buffer's text with DOS-style carriage return followed by linefeed line endings.

Another way to specify the coding system for a file is when you visit the file. First use the command `C-x RET c` (`universal-coding-system-argument`); this command uses the minibuffer to read a coding system name. After you exit the minibuffer, the specified coding system is used for *the immediately following command*.

So if the immediately following command is `C-x C-f`, for example, it reads the file using that coding system (and records the coding system for when you later save the file). Or if the immediately following command is `C-x C-w`, it writes the file using that coding system. When you specify the coding system for saving in this way, instead of with `C-x RET f`, there is no warning if the buffer contains characters that the coding system cannot handle.

Other file commands affected by a specified coding system include `C-x i` and `C-x C-v`, as well as the other-window variants of `C-x C-f`. `C-x RET c` also affects commands that start subprocesses, including `M-x shell` (see Section 31.5 [Shell], page 423). If the immediately following command does not use the coding system, then `C-x RET c` ultimately has no effect.

An easy way to visit a file with no conversion is with the `M-x find-file-literally` command. See Section 15.2 [Visiting], page 136.

The default value of the variable `buffer-file-coding-system` specifies the choice of coding system to use when you create a new file. It applies when you find a new file, and when you create a buffer and then save it in a file. Selecting a language environment typically sets this variable to a good choice of default coding system for that language environment.

If you visit a file with a wrong coding system, you can correct this with `C-x RET r` (`revert-buffer-with-coding-system`). This visits the current file again, using a coding system you specify.

If a piece of text has already been inserted into a buffer using the wrong coding system, you can redo the decoding of it using `M-x recode-region`. This prompts you for the proper coding system, then for the wrong coding system that was actually used, and does the conversion. It first encodes the region using the wrong coding system, then decodes it again using the proper coding system.

19.10 Coding Systems for Interprocess Communication

This section explains how to specify coding systems for use in communication with other processes.

`C-x RET x` *coding* `RET`
: Use coding system *coding* for transferring selections to and from other graphical applications (`set-selection-coding-system`).

`C-x RET X` *coding* `RET`
: Use coding system *coding* for transferring *one* selection—the next one—to or from another graphical application (`set-next-selection-coding-system`).

`C-x RET p` *input-coding* `RET` *output-coding* `RET`
: Use coding systems *input-coding* and *output-coding* for subprocess input and output in the current buffer (`set-buffer-process-coding-system`).

The command `C-x RET x` (`set-selection-coding-system`) specifies the coding system for sending selected text to other windowing applications, and for receiving the text of selections made in other applications. This command applies to all subsequent selections, until you override it by using the command again. The command `C-x RET X` (`set-next-selection-coding-system`) specifies the coding system for the next selection made in Emacs or read by Emacs.

The variable `x-select-request-type` specifies the data type to request from the X Window System for receiving text selections from other applications. If the value is `nil` (the default), Emacs tries `UTF8_STRING` and `COMPOUND_TEXT`, in this order, and uses various heuristics to choose the more appropriate of the two results; if none of these succeed, Emacs falls back on `STRING`. If the value of `x-select-request-type` is one of the symbols `COMPOUND_TEXT`, `UTF8_STRING`, `STRING`, or `TEXT`, Emacs uses only that request type. If the value is a list of some of these symbols, Emacs tries only the request types in the list, in order, until one of them succeeds, or until the list is exhausted.

The command `C-x RET p` (`set-buffer-process-coding-system`) specifies the coding system for input and output to a subprocess. This command applies to the current buffer; normally, each subprocess has its own buffer, and thus you can use this command to specify translation to and from a particular subprocess by giving the command in the corresponding buffer.

You can also use `C-x RET c` (`universal-coding-system-argument`) just before the command that runs or starts a subprocess, to specify the coding system for communicating with that subprocess. See Section 19.9 [Text Coding], page 212.

The default for translation of process input and output depends on the current language environment.

The variable `locale-coding-system` specifies a coding system to use when encoding and decoding system strings such as system error messages and `format-time-string` formats and time stamps. That coding system is also used for decoding non-ASCII keyboard input on the X Window System and for encoding text sent to the standard output and error streams when in batch mode. You should choose a coding system that is compatible with the underlying system's text representation, which is normally specified by one of the environment variables `LC_ALL`, `LC_CTYPE`, and `LANG`. (The first one, in the order specified above, whose value is nonempty is the one that determines the text representation.)

19.11 Coding Systems for File Names

`C-x RET F` *coding* `RET`
: Use coding system *coding* for encoding and decoding file names (`set-file-name-coding-system`).

The command `C-x RET F` (`set-file-name-coding-system`) specifies a coding system to use for encoding file *names*. It has no effect on reading and writing the *contents* of files.

In fact, all this command does is set the value of the variable `file-name-coding-system`. If you set the variable to a coding system name (as a Lisp symbol or a string), Emacs encodes file names using that coding system for all file operations. This makes it possible to use non-ASCII characters in file names—or, at least, those non-ASCII characters that the specified coding system can encode.

If `file-name-coding-system` is `nil`, Emacs uses a default coding system determined by the selected language environment, and stored in the `default-file-name-coding-system` variable. In the default language environment, non-ASCII characters in file names are not encoded specially; they appear in the file system using the internal Emacs representation.

When Emacs runs on MS-Windows versions that are descendants of the NT family (Windows 2000, XP, and all the later versions), the value of `file-name-coding-system` is largely ignored, as Emacs by default uses APIs that allow passing Unicode file names directly. By contrast, on Windows 9X, file names are encoded using `file-name-coding-system`, which should be set to the codepage (see Section 19.5 [Coding Systems], page 208) pertinent for the current system locale. The value of the variable `w32-unicode-filenames` controls whether Emacs uses the Unicode APIs when it calls OS functions that accept file names. This variable is set by the startup code to `nil` on Windows 9X, and to `t` on newer versions of MS-Windows.

Warning: if you change `file-name-coding-system` (or the language environment) in the middle of an Emacs session, problems can result if you have already visited files whose names were encoded using the earlier coding system and cannot be encoded (or are encoded differently) under the new coding system. If you try to save one of these buffers under the visited file name, saving may use the wrong file name, or it may encounter an error. If such a problem happens, use `C-x C-w` to specify a new file name for that buffer.

If a mistake occurs when encoding a file name, use the command `M-x recode-file-name` to change the file name's coding system. This prompts for an existing file name, its old coding system, and the coding system to which you wish to convert.

19.12 Coding Systems for Terminal I/O

`C-x RET t` *coding* `RET`
: Use coding system *coding* for terminal output (`set-terminal-coding-system`).

`C-x RET k` *coding* `RET`
: Use coding system *coding* for keyboard input (`set-keyboard-coding-system`).

The command `C-x RET t` (`set-terminal-coding-system`) specifies the coding system for terminal output. If you specify a character code for terminal output, all characters output to the terminal are translated into that coding system.

This feature is useful for certain character-only terminals built to support specific languages or character sets—for example, European terminals that support one of the ISO Latin character sets. You need to specify the terminal coding system when using multibyte text, so that Emacs knows which characters the terminal can actually handle.

By default, output to the terminal is not translated at all, unless Emacs can deduce the proper coding system from your terminal type or your locale specification (see Section 19.2 [Language Environments], page 203).

The command `C-x RET k` (`set-keyboard-coding-system`), or the variable `keyboard-coding-system`, specifies the coding system for keyboard input. Character-code translation of keyboard input is useful for terminals with keys that send non-ASCII graphic characters—for example, some terminals designed for ISO Latin-1 or subsets of it.

By default, keyboard input is translated based on your system locale setting. If your terminal does not really support the encoding implied by your locale (for example, if you find it inserts a non-ASCII character if you type `M-i`), you will need to set `keyboard-coding-system` to `nil` to turn off encoding. You can do this by putting

```
(set-keyboard-coding-system nil)
```

in your init file.

There is a similarity between using a coding system translation for keyboard input, and using an input method: both define sequences of keyboard input that translate into single characters. However, input methods are designed to be convenient for interactive use by humans, and the sequences that are translated are typically sequences of ASCII printing characters. Coding systems typically translate sequences of non-graphic characters.

19.13 Fontsets

A font typically defines shapes for a single alphabet or script. Therefore, displaying the entire range of scripts that Emacs supports requires a collection of many fonts. In Emacs, such a collection is called a *fontset*. A fontset is defined by a list of font specifications, each assigned to handle a range of character codes, and may fall back on another fontset for characters that are not covered by the fonts it specifies.

Each fontset has a name, like a font. However, while fonts are stored in the system and the available font names are defined by the system, fontsets are defined within Emacs itself. Once you have defined a fontset, you can use it within Emacs by specifying its name, anywhere that you could use a single font. Of course, Emacs fontsets can use only the fonts that your system supports. If some characters appear on the screen as empty boxes or hex codes, this means that the fontset in use for them has no font for those characters. In this case, or if the characters are shown, but not as well as you would like, you may need to install extra fonts or modify the fontset to use specific fonts already installed on your system (see below). Your operating system may have optional fonts that you can install; or you can install the GNU Intlfonts package, which includes fonts for most supported scripts.[2]

Emacs creates three fontsets automatically: the *standard fontset*, the *startup fontset* and the *default fontset*. The default fontset is most likely to have fonts for a wide variety of non-ASCII characters, and is the default fallback for the other two fontsets, and if you set a default font rather than fontset. However, it does not specify font family names, so

[2] If you run Emacs on X, you may need to inform the X server about the location of the newly installed fonts with commands such as:

```
xset fp+ /usr/local/share/emacs/fonts
xset fp rehash
```

results can be somewhat random if you use it directly. You can specify a particular fontset by starting Emacs with the '`-fn`' option. For example,

```
emacs -fn fontset-standard
```

You can also specify a fontset with the '`Font`' resource (see Appendix D [X Resources], page 547).

If no fontset is specified for use, then Emacs uses an ASCII font, with '`fontset-default`' as a fallback for characters the font does not cover. The standard fontset is only used if explicitly requested, despite its name.

To show the information about a specific fontset, use the `M-x describe-fontset` command. It prompts for a fontset name, defaulting to the one used by the current frame, and then displays all the subranges of characters and the fonts assigned to them in that fontset. To see which fonts Emacs is using in a session started without a specific fontset (which is what happens normally), type `fontset-default RET` at the prompt, or just `RET` to describe the fontset used by the current frame.

A fontset does not necessarily specify a font for every character code. If a fontset specifies no font for a certain character, or if it specifies a font that does not exist on your system, then it cannot display that character properly. It will display that character as a hex code or thin space or an empty box instead. (See Section 11.19 [glyphless characters], page 91, for details.) Or a fontset might specify a font for some range of characters, but you may not like their visual appearance. If this happens, you may wish to modify your fontset; see Section 19.15 [Modifying Fontsets], page 219, for how to do that.

19.14 Defining Fontsets

When running on X, Emacs creates a standard fontset automatically according to the value of `standard-fontset-spec`. This fontset's name is

```
-*-fixed-medium-r-normal-*-16-*-*-*-*-*-fontset-standard
```

or just '`fontset-standard`' for short.

On GNUstep and macOS, the standard fontset is created using the value of `ns-standard-fontset-spec`, and on MS Windows it is created using the value of `w32-standard-fontset-spec`.

Bold, italic, and bold-italic variants of the standard fontset are created automatically. Their names have '`bold`' instead of '`medium`', or '`i`' instead of '`r`', or both.

Emacs generates a fontset automatically, based on any default ASCII font that you specify with the '`Font`' resource or the '`-fn`' argument, or the default font that Emacs found when it started. This is the *startup fontset* and its name is `fontset-startup`. Emacs generates this fontset by replacing the *charset_registry* field with '`fontset`', and replacing the *charset_encoding* field with '`startup`', then using the resulting string to specify a fontset.

For instance, if you start Emacs with a font of this form,

```
emacs -fn "*courier-medium-r-normal--14-140-*-iso8859-1"
```

Emacs generates the following fontset and uses it for the initial X window frame:

```
-*-courier-medium-r-normal-*-14-140-*-*-*-*-fontset-startup
```

The startup fontset will use the font that you specify, or a variant with a different registry and encoding, for all the characters that are supported by that font, and fallback on '`fontset-default`' for other characters.

With the X resource '`Emacs.Font`', you can specify a fontset name just like an actual font name. But be careful not to specify a fontset name in a wildcard resource like '`Emacs*Font`'—that wildcard specification matches various other resources, such as for menus, and menus cannot handle fontsets. See Appendix D [X Resources], page 547.

You can specify additional fontsets using X resources named '`Fontset-`*n*', where *n* is an integer starting from 0. The resource value should have this form:

fontpattern, [*charset*:*font*]...

where *fontpattern* should have the form of a standard X font name (see the previous fontset-startup example), except for the last two fields. They should have the form '`fontset-`*alias*'.

Each fontset has two names, one long and one short. The long name is *fontpattern*. The short name is '`fontset-`*alias*', the last 2 fields of the long name (e.g., '`fontset-startup`' for the fontset automatically created at startup). You can refer to the fontset by either name.

The construct '*charset*:*font*' specifies which font to use (in this fontset) for one particular character set. Here, *charset* is the name of a character set, and *font* is the font to use for that character set. You can use this construct any number of times in defining one fontset.

For the other character sets, Emacs chooses a font based on *fontpattern*. It replaces '`fontset-`*alias*' with values that describe the character set. For the ASCII character font, '`fontset-`*alias*' is replaced with '`ISO8859-1`'.

In addition, when several consecutive fields are wildcards, Emacs collapses them into a single wildcard. This is to prevent use of auto-scaled fonts. Fonts made by scaling larger fonts are not usable for editing, and scaling a smaller font is also not useful, because it is better to use the smaller font in its own size, which is what Emacs does.

Thus if *fontpattern* is this,

```
-*-fixed-medium-r-normal-*-24-*-*-*-*-*-fontset-24
```

the font specification for ASCII characters would be this:

```
-*-fixed-medium-r-normal-*-24-*-ISO8859-1
```

and the font specification for Chinese GB2312 characters would be this:

```
-*-fixed-medium-r-normal-*-24-*-gb2312*-*
```

You may not have any Chinese font matching the above font specification. Most X distributions include only Chinese fonts that have '`song ti`' or '`fangsong ti`' in the *family* field. In such a case, '`Fontset-`*n*' can be specified as:

```
Emacs.Fontset-0: -*-fixed-medium-r-normal-*-24-*-*-*-*-*-fontset-24,\
        chinese-gb2312:-*-*-medium-r-normal-*-24-*-gb2312*-*
```

Then, the font specifications for all but Chinese GB2312 characters have '`fixed`' in the *family* field, and the font specification for Chinese GB2312 characters has a wild card '`*`' in the *family* field.

The function that processes the fontset resource value to create the fontset is called `create-fontset-from-fontset-spec`. You can also call this function explicitly to create a fontset.

See Section 18.8 [Fonts], page 187, for more information about font naming.

19.15 Modifying Fontsets

Fontsets do not always have to be created from scratch. If only minor changes are required it may be easier to modify an existing fontset, usually '`fontset-default`'. Modifying '`fontset-default`' will also affect other fontsets that use it as a fallback, so can be an effective way of fixing problems with the fonts that Emacs chooses for a particular script.

Fontsets can be modified using the function `set-fontset-font`, specifying a character, a charset, a script, or a range of characters to modify the font for, and a font specification for the font to be used. Some examples are:

```
;; Prefer a big5 font for han characters.
(set-fontset-font "fontset-default"
                  'han (font-spec :registry "big5")
                  nil 'prepend)

;; Use MyPrivateFont for the Unicode private use area.
(set-fontset-font "fontset-default"  '(#xe000 . #xf8ff)
                  "MyPrivateFont")

;; Use Liberation Mono for latin-3 charset.
(set-fontset-font "fontset-default" 'iso-8859-3
                  "Liberation Mono")

;; Use DejaVu Sans Mono as a fallback in fontset-startup
;; before resorting to fontset-default.
(set-fontset-font "fontset-startup" nil "DejaVu Sans Mono"
                  nil 'append)
```

See Section "Fontsets" in *GNU Emacs Lisp Reference Manual*, for more details about using the `set-fontset-font` function.

If you don't know the character's codepoint or the script to which it belongs, you can ask Emacs. With point at the character, type `C-u C-x =` (`what-cursor-position`), and this information, together with much more, will be displayed in the `*Help*` buffer that Emacs pops up. See Section 4.9 [Position Info], page 22. For example, Japanese characters belong to the '`kana`' script, but Japanese text also mixes them with Chinese characters so the following uses the '`han`' script to set up Emacs to use the '`Kochi Gothic`' font for Japanese text:

```
(set-fontset-font "fontset-default" 'han "Kochi Gothic")
```

(For convenience, the '`han`' script in Emacs is set up to support all of the Chinese, Japanese, and Korean, a.k.a. CJK, characters, not just Chinese characters.)

For the list of known scripts, see the variable `script-representative-chars`.

Fontset settings like those above only affect characters that the default font doesn't support, so if the '`Kochi Gothic`' font covers Latin characters, it will not be used for displaying Latin scripts, since the default font used by Emacs usually covers Basic Latin.

Some fonts installed on your system might be broken, or produce unpleasant results for characters for which they are used, and you may wish to instruct Emacs to completely ignore them while searching for a suitable font required to display a character. You can do

that by adding the offending fonts to the value of the variable `face-ignored-fonts`, which is a list. Here's an example to put in your `~/.emacs`:

```
(add-to-list 'face-ignored-fonts "Some Bad Font")
```

19.16 Undisplayable Characters

There may be some non-ASCII characters that your terminal cannot display. Most text terminals support just a single character set (use the variable `default-terminal-coding-system` to tell Emacs which one, Section 19.12 [Terminal Coding], page 215); characters that can't be encoded in that coding system are displayed as '`?`' by default.

Graphical displays can display a broader range of characters, but you may not have fonts installed for all of them; characters that have no font appear as a hollow box.

If you use Latin-1 characters but your terminal can't display Latin-1, you can arrange to display mnemonic ASCII sequences instead, e.g., '`"o`' for o-umlaut. Load the library `iso-ascii` to do this.

If your terminal can display Latin-1, you can display characters from other European character sets using a mixture of equivalent Latin-1 characters and ASCII mnemonics. Customize the variable `latin1-display` to enable this. The mnemonic ASCII sequences mostly correspond to those of the prefix input methods.

19.17 Unibyte Editing Mode

The ISO 8859 Latin-*n* character sets define character codes in the range 0240 to 0377 octal (160 to 255 decimal) to handle the accented letters and punctuation needed by various European languages (and some non-European ones). Note that Emacs considers bytes with codes in this range as raw bytes, not as characters, even in a unibyte buffer, i.e., if you disable multibyte characters. However, Emacs can still handle these character codes as if they belonged to *one* of the single-byte character sets at a time. To specify *which* of these codes to use, invoke `M-x set-language-environment` and specify a suitable language environment such as '`Latin-n`'. See Section "Disabling Multibyte Characters" in *GNU Emacs Lisp Reference Manual*.

Emacs can also display bytes in the range 160 to 255 as readable characters, provided the terminal or font in use supports them. This works automatically. On a graphical display, Emacs can also display single-byte characters through fontsets, in effect by displaying the equivalent multibyte characters according to the current language environment. To request this, set the variable `unibyte-display-via-language-environment` to a non-`nil` value. Note that setting this only affects how these bytes are displayed, but does not change the fundamental fact that Emacs treats them as raw bytes, not as characters.

If your terminal does not support display of the Latin-1 character set, Emacs can display these characters as ASCII sequences which at least give you a clear idea of what the characters are. To do this, load the library `iso-ascii`. Similar libraries for other Latin-*n* character sets could be implemented, but have not been so far.

Normally non-ISO-8859 characters (decimal codes between 128 and 159 inclusive) are displayed as octal escapes. You can change this for non-standard extended versions of ISO-8859 character sets by using the function `standard-display-8bit` in the `disp-table` library.

There are two ways to input single-byte non-ASCII characters:

- You can use an input method for the selected language environment. See Section 19.3 [Input Methods], page 205. When you use an input method in a unibyte buffer, the non-ASCII character you specify with it is converted to unibyte.
- If your keyboard can generate character codes 128 (decimal) and up, representing non-ASCII characters, you can type those character codes directly.

  On a graphical display, you should not need to do anything special to use these keys; they should simply work. On a text terminal, you should use the command `M-x set-keyboard-coding-system` or customize the variable `keyboard-coding-system` to specify which coding system your keyboard uses (see Section 19.12 [Terminal Coding], page 215). Enabling this feature will probably require you to use `ESC` to type Meta characters; however, on a console terminal or a terminal emulator such as `xterm`, you can arrange for Meta to be converted to `ESC` and still be able to type 8-bit characters present directly on the keyboard or using `Compose` or `AltGr` keys. See Section 2.1 [User Input], page 11.
- You can use the key `C-x 8` as a compose-character prefix for entry of non-ASCII Latin-1 and a few other printing characters. `C-x 8` is good for insertion (in the minibuffer as well as other buffers), for searching, and in any other context where a key sequence is allowed.

  `C-x 8` works by loading the `iso-transl` library. Once that library is loaded, the `Alt` modifier key, if the keyboard has one, serves the same purpose as `C-x 8`: use `Alt` together with an accent character to modify the following letter. In addition, if the keyboard has keys for the Latin-1 dead accent characters, they too are defined to compose with the following character, once `iso-transl` is loaded.

  Use `C-x 8 C-h` to list all the available `C-x 8` translations.

19.18 Charsets

In Emacs, *charset* is short for "character set". Emacs supports most popular charsets (such as `ascii`, `iso-8859-1`, `cp1250`, `big5`, and `unicode`), in addition to some charsets of its own (such as `emacs`, `unicode-bmp`, and `eight-bit`). All supported characters belong to one or more charsets.

Emacs normally does the right thing with respect to charsets, so that you don't have to worry about them. However, it is sometimes helpful to know some of the underlying details about charsets.

One example is font selection (see Section 18.8 [Fonts], page 187). Each language environment (see Section 19.2 [Language Environments], page 203) defines a priority list for the various charsets. When searching for a font, Emacs initially attempts to find one that can display the highest-priority charsets. For instance, in the Japanese language environment, the charset `japanese-jisx0208` has the highest priority, so Emacs tries to use a font whose `registry` property is '`JISX0208.1983-0`'.

There are two commands that can be used to obtain information about charsets. The command `M-x list-charset-chars` prompts for a charset name, and displays all the characters in that character set. The command `M-x describe-character-set` prompts for a charset name, and displays information about that charset, including its internal representation within Emacs.

`M-x list-character-sets` displays a list of all supported charsets. The list gives the names of charsets and additional information to identity each charset; for more details, see the ISO International Register of Coded Character Sets to be Used with Escape Sequences (ISO-IR) (`https://www.itscj.ipsj.or.jp/itscj_english/iso-ir/ISO-IR.pdf`) maintained by the Information Processing Society of Japan/Information Technology Standards Commission of Japan (IPSJ/ITSCJ) (`https://www.itscj.ipsj.or.jp/itscj_english/`). In this list, charsets are divided into two categories: *normal charsets* are listed first, followed by *supplementary charsets*. A supplementary charset is one that is used to define another charset (as a parent or a subset), or to provide backward-compatibility for older Emacs versions.

To find out which charset a character in the buffer belongs to, put point before it and type `C-u C-x =` (see Section 19.1 [International Chars], page 201).

19.19 Bidirectional Editing

Emacs supports editing text written in scripts, such as Arabic, Farsi, and Hebrew, whose natural ordering of horizontal text for display is from right to left. However, digits and Latin text embedded in these scripts are still displayed left to right. It is also not uncommon to have small portions of text in Arabic or Hebrew embedded in an otherwise Latin document; e.g., as comments and strings in a program source file. For these reasons, text that uses these scripts is actually *bidirectional*: a mixture of runs of left-to-right and right-to-left characters.

This section describes the facilities and options provided by Emacs for editing bidirectional text.

Emacs stores right-to-left and bidirectional text in the so-called *logical* (or *reading*) order: the buffer or string position of the first character you read precedes that of the next character. Reordering of bidirectional text into the *visual* order happens at display time. As a result, character positions no longer increase monotonically with their positions on display. Emacs implements the Unicode Bidirectional Algorithm (UBA) described in the Unicode Standard Annex #9 (`https://unicode.org/reports/tr9/`), for reordering of bidirectional text for display. It deviates from the UBA only in how continuation lines are displayed when text direction is opposite to the base paragraph direction, e.g., when a long line of English text appears in a right-to-left paragraph.

The buffer-local variable `bidi-display-reordering` controls whether text in the buffer is reordered for display. If its value is non-`nil`, Emacs reorders characters that have right-to-left directionality when they are displayed. The default value is `t`.

Each paragraph of bidirectional text can have its own *base direction*, either right-to-left or left-to-right. Text in left-to-right paragraphs begins on the screen at the left margin of the window and is truncated or continued when it reaches the right margin. By contrast, text in right-to-left paragraphs is displayed starting at the right margin and is continued or truncated at the left margin. By default, paragraph boundaries are empty lines, i.e., lines consisting entirely of whitespace characters. To change that, you can customize the two variables `bidi-paragraph-start-re` and `bidi-paragraph-separate-re`, whose values should be regular expressions (strings); e.g., to have a single newline start a new paragraph, set both of these variables to `"^"`. These two variables are buffer-local (see Section 33.2.3 [Locals], page 470).

Emacs determines the base direction of each paragraph dynamically, based on the text at the beginning of the paragraph. However, sometimes a buffer may need to force a certain base direction for its paragraphs. The variable `bidi-paragraph-direction`, if non-`nil`, disables the dynamic determination of the base direction, and instead forces all paragraphs in the buffer to have the direction specified by its buffer-local value. The value can be either `right-to-left` or `left-to-right`. Any other value is interpreted as `nil`.

Alternatively, you can control the base direction of a paragraph by inserting special formatting characters in front of the paragraph. The special character `RIGHT-TO-LEFT MARK`, or RLM, forces the right-to-left direction on the following paragraph, while `LEFT-TO-RIGHT MARK`, or LRM forces the left-to-right direction. (You can use `C-x 8 RET` to insert these characters.) In a GUI session, the LRM and RLM characters display as very thin blank characters; on text terminals they display as blanks.

Because characters are reordered for display, Emacs commands that operate in the logical order or on stretches of buffer positions may produce unusual effects. For example, the commands `C-f` and `C-b` move point in the logical order, so the cursor will sometimes jump when point traverses reordered bidirectional text. Similarly, a highlighted region covering a contiguous range of character positions may look discontinuous if the region spans reordered text. This is normal and similar to the behavior of other programs that support bidirectional text.

Cursor motion commands bound to arrow keys, such as `LEFT` and `C-RIGHT`, are sensitive to the base direction of the current paragraph. In a left-to-right paragraph, commands bound to `RIGHT` with or without modifiers move *forward* through buffer text, but in a right-to-left paragraph they move *backward* instead. This reflects the fact that in a right-to-left paragraph buffer positions predominantly increase when moving to the left on display.

When you move out of a paragraph, the meaning of the arrow keys might change if the base direction of the preceding or the following paragraph is different from the paragraph out of which you moved. When that happens, you need to adjust the arrow key you press to the new base direction.

By default, `LEFT` and `RIGHT` move in the logical order, but if `visual-order-cursor-movement` is non-`nil`, these commands move to the character that is, correspondingly, to the left or right of the current screen position, moving to the next or previous screen line as appropriate. Note that this might potentially move point many buffer positions away, depending on the surrounding bidirectional context.