```
Special issues ~
|print.txt| printing
|remote.txt|    using Vim as a server or client
|term.txt|  using different terminals and mice
|digraph.txt|   list of available digraphs
|mbyte.txt| multi-byte text support
|mlang.txt| non-English language support
|arabic.txt|    Arabic language support and editing
|farsi.txt| Farsi (Persian) editing
|hebrew.txt|    Hebrew language support and editing
|russian.txt|   Russian language support and editing
|ft_ada.txt|    Ada (the programming language) support
|ft_sql.txt|    about the SQL filetype plugin
|hangulin.txt|  Hangul (Korean) input mode
|rileft.txt|    right-to-left editing mode

==============================================================================
*print.txt*     For Vim version 8.0.  Last change: 2010 Jul 20


                  VIM REFERENCE MANUAL    by Bram Moolenaar


Printing                                                *printing*

1. Introduction                         |print-intro|
2. Print options                        |print-options|
3. PostScript Printing                  |postscript-printing|
4. PostScript Printing Encoding         |postscript-print-encoding|
5. PostScript CJK Printing              |postscript-cjk-printing|
6. PostScript Printing Troubleshooting  |postscript-print-trouble|
7. PostScript Utilities                 |postscript-print-util|
8. Formfeed Characters                  |printing-formfeed|

{Vi has None of this}
{only available when compiled with the |+printer| feature}

==============================================================================
1. Introduction                                         *print-intro*

On MS-Windows Vim can print your text on any installed printer.  On other
systems a PostScript file is produced.  This can be directly sent to a
PostScript printer.  For other printers a program like ghostscript needs to be
used.

Note: If you have problems printing with |:hardcopy|, an alternative is to use
|:TOhtml| and print the resulting html file from a browser.

                                        *:ha* *:hardcopy* *E237* *E238* *E324*
:[range]ha[rdcopy][!] [arguments]
                        Send [range] lines (default whole file) to the
                        printer.

                        On MS-Windows a dialog is displayed to allow selection
                        of printer, paper size etc.  To skip the dialog, use
                        the [!].  In this case the printer defined by
                        'printdevice' is used, or, if 'printdevice' is empty,
                        the system default printer.

                        For systems other than MS-Windows, PostScript is
                        written in a temp file and 'printexpr' is used to
                        actually print it.  Then [arguments] can be used by
                        'printexpr' through |v:cmdarg|.  Otherwise [arguments]
                        is ignored.  'printoptions' can be used to specify
                        paper size, duplex, etc.

:[range]ha[rdcopy][!] >{filename}
                        As above, but write the resulting PostScript in file
                        {filename}.
                        Things like "%" are expanded |cmdline-special|
                        Careful: An existing file is silently overwritten.
                        {only available when compiled with the |+postscript|
                        feature}
                        On MS-Windows use the "print to file" feature of the
                        printer driver.

Progress is displayed during printing as a page number and a percentage.  To
abort printing use the interrupt key (CTRL-C or, on MS-systems, CTRL-Break).

Printer output is controlled by the 'printfont' and 'printoptions' options.
'printheader' specifies the format of a page header.

The printed file is always limited to the selected margins, irrespective of
the current window's 'wrap' or 'linebreak' settings.  The "wrap" item in
'printoptions' can be used to switch wrapping off.
The current highlighting colors are used in the printout, with the following
considerations:
1) The normal background is always rendered as white (i.e. blank paper).
2) White text or the default foreground is rendered as black, so that it shows
   up!
3) If 'background' is "dark", then the colours are darkened to compensate for
   the fact that otherwise they would be too bright to show up clearly on
   white paper.

==============================================================================
2. Print options                                        *print-options*

Here are the details for the options that change the way printing is done.
For generic info about setting options see |options.txt|.

                                                        *pdev-option*
'printdevice' 'pdev'    string  (default empty)
                        global
This defines the name of the printer to be used when the |:hardcopy| command
is issued with a bang (!) to skip the printer selection dialog.  On Win32, it
should be the printer name exactly as it appears in the standard printer
dialog.
If the option is empty, then vim will use the system default printer for
":hardcopy!"

                                                        *penc-option* *E620*
'printencoding' 'penc'  String  (default empty, except for:
                                        Windows, OS/2: cp1252,
                                        Macintosh: mac-roman,
                                        VMS: dec-mcs,
                                        HPUX: hp-roman8,
                                        EBCDIC: ebcdic-uk)
                        global
Sets the character encoding used when printing.  This option tells Vim which
print character encoding file from the "print" directory in 'runtimepath' to
use.

This option will accept any value from |encoding-names|.  Any recognized names
are converted to Vim standard names - see 'encoding' for more details.  Names
```

```
not recognized by Vim will just be converted to lower case and underscores
replaced with '-' signs.

If 'printencoding' is empty or Vim cannot find the file then it will use
'encoding' (if Vim is compiled with |+multi_byte| and it is set an 8-bit
encoding) to find the print character encoding file.  If Vim is unable to find
a character encoding file then it will use the "latin1" print character
encoding file.

When 'encoding' is set to a multi-byte encoding, Vim will try to convert
characters to the printing encoding for printing (if 'printencoding' is empty
then the conversion will be to latin1).  Conversion to a printing encoding
other than latin1 will require Vim to be compiled with the |+iconv| feature.
If no conversion is possible then printing will fail.  Any characters that
cannot be converted will be replaced with upside down question marks.

Four print character encoding files are provided to support default Mac, VMS,
HPUX, and EBCDIC character encodings and are used by default on these
platforms.  Code page 1252 print character encoding is used by default on
Windows and OS/2 platforms.

                                                        *pexpr-option*
'printexpr' 'pexpr'     String  (default: see below)
                        global
Expression that is evaluated to print the PostScript produced with
|:hardcopy|.
The file name to be printed is in |v:fname_in|.
The arguments to the ":hardcopy" command are in |v:cmdarg|.
The expression must take care of deleting the file after printing it.
When there is an error, the expression must return a non-zero number.
If there is no error, return zero or an empty string.
The default for non MS-Windows or VMS systems is to simply use "lpr" to print
the file: >

    system('lpr' . (&printdevice == '' ? '' : ' -P' . &printdevice)
        . ' ' . v:fname_in) . delete(v:fname_in) + v:shell_error

On MS-Dos, MS-Windows and OS/2 machines the default is to copy the file to the
currently specified printdevice: >

    system('copy' . ' ' . v:fname_in . (&printdevice == ''
                ? ' LPT1:' : (' \"' . &printdevice . '\"')))
                . delete(v:fname_in)

On VMS machines the default is to send the file to either the default or
currently specified printdevice: >

    system('print' . (&printdevice == '' ? '' : ' /queue=' .
                &printdevice) . ' ' . v:fname_in) . delete(v:fname_in)

If you change this option, using a function is an easy way to avoid having to
escape all the spaces.  Example: >

        :set printexpr=PrintFile(v:fname_in)
        :function PrintFile(fname)
        :  call system("ghostview " . a:fname)
        :  call delete(a:fname)
        :  return v:shell_error
        :endfunc

Be aware that some print programs return control before they have read the
file.  If you delete the file too soon it will not be printed.  These programs
```

usually offer an option to have them remove the file when printing is done.

```
                                                        *E365*
```

If evaluating the expression fails or it results in a non-zero number, you get
an error message. In that case Vim will delete the file. In the default
value for non-MS-Windows a trick is used: Adding "v:shell_error" will result
in a non-zero number when the system() call fails.

This option cannot be set from a |modeline| or in the |sandbox|, for security
reasons.

```
                                                        *pfn-option* *E613*
'printfont' 'pfn'       string  (default "courier")
                        global
```

This is the name of the font that will be used for the |:hardcopy| command's
output. It has the same format as the 'guifont' option, except that only one
font may be named, and the special "guifont=*" syntax is not available.

In the Win32 GUI version this specifies a font name with its extra attributes,
as with the 'guifont' option.

For other systems, only ":h11" is recognized, where "11" is the point size of
the font. When omitted, the point size is 10.

```
                                                        *pheader-option*
'printheader' 'pheader'  string  (default "%<%f%h%m%=Page %N")
                         global
```

This defines the format of the header produced in |:hardcopy| output. The
option is defined in the same way as the 'statusline' option. If Vim has not
been compiled with the |+statusline| feature, this option has no effect and a
simple default header is used, which shows the page number. The same simple
header is used when this option is empty.

```
                                                        *pmbcs-option*
'printmbcharset' 'pmbcs'  string (default "")
                          global
```

Sets the CJK character set to be used when generating CJK output from
|:hardcopy|. The following predefined values are currently recognised by Vim:

	Value	Description ~
Chinese	GB_2312-80	
(Simplified)	GBT_12345-90	
	MAC	Apple Mac Simplified Chinese
	GBT-90_MAC	GB/T 12345-90 Apple Mac Simplified Chinese
	GBK	GBK (GB 13000.1-93)
	ISO10646	ISO 10646-1:1993
Chinese	CNS_1993	CNS 11643-1993, Planes 1 & 2
(Traditional)	BIG5	
	ETEN	Big5 with ETen extensions
	ISO10646	ISO 10646-1:1993
Japanese	JIS_C_1978	
	JIS_X_1983	
	JIS_X_1990	
	MSWINDOWS	Win3.1/95J (JIS X 1997 + NEC + IBM extensions)
	KANJITALK6	Apple Mac KanjiTalk V6.x
	KANJITALK7	Apple Mac KanjiTalk V7.x
Korean	KS_X_1992	
	MAC	Apple Macintosh Korean

MSWINDOWS	KS X 1992 with MS extensions
ISO10646	ISO 10646-1:1993

Only certain combinations of the above values and 'printencoding' are
possible. The following tables show the valid combinations:

		euc-cn	gbk	ucs-2	utf-8 ~
Chinese	GB_2312-80	x			
(Simplified)	GBT_12345-90	x			
	MAC	x			
	GBT-90_MAC	x			
	GBK		x		
	ISO10646			x	x

		euc-tw	big5	ucs-2	utf-8 ~
Chinese	CNS_1993	x			
(Traditional)	BIG5		x		
	ETEN		x		
	ISO10646			x	x

		euc-jp	sjis	ucs-2	utf-8 ~
Japanese	JIS_C_1978	x	x		
	JIS_X_1983	x	x		
	JIS_X_1990	x		x	x
	MSWINDOWS	x			
	KANJITALK6	x			
	KANJITALK7	x			

		euc-kr	cp949	ucs-2	utf-8 ~
Korean	KS_X_1992	x			
	MAC	x			
	MSWINDOWS		x		
	ISO10646			x	x

To set up the correct encoding and character set for printing some
Japanese text you would do the following; >

```
        :set printencoding=euc-jp
        :set printmbcharset=JIS_X_1983
```

If 'printmbcharset' is not one of the above values then it is assumed to
specify a custom multi-byte character set and no check will be made that it is
compatible with the value for 'printencoding'. Vim will look for a file
defining the character set in the "print" directory in 'runtimepath'.

pmbfn-option

'printmbfont' 'pmbfn' string (default "")
global

This is a comma-separated list of fields for font names to be used when
generating CJK output from |:hardcopy|. Each font name has to be preceded
with a letter indicating the style the font is to be used for as follows:

r:{font-name}	font to use for normal characters
b:{font-name}	font to use for bold characters
i:{font-name}	font to use for italic characters
o:{font-name}	font to use for bold-italic characters

A field with the r: prefix must be specified when doing CJK printing. The
other fontname specifiers are optional. If a specifier is missing then
another font will be used as follows:

if b: is missing, then use r:
if i: is missing, then use r:

```
  if o: is missing, then use b:

Some CJK fonts do not contain characters for codes in the ASCII code range.
Also, some characters in the CJK ASCII code ranges differ in a few code points
from traditional ASCII characters.  There are two additional fields to control
printing of characters in the ASCII code range.

  c:yes                 Use Courier font for characters in the ASCII
  c:no (default)        code range.

  a:yes                 Use ASCII character set for codes in the ASCII
  a:no (default)        code range.

The following is an example of specifying two multi-byte fonts, one for normal
and italic printing and one for bold and bold-italic printing, and using
Courier to print codes in the ASCII code range but using the national
character set: >
        :set printmbfont=r:WadaMin-Regular,b:WadaMin-Bold,c:yes
<
                                                        *popt-option*
'printoptions' 'popt'   string (default "")
                        global
This is a comma-separated list of items that control the format of the output
of |:hardcopy|:

  left:{spec}           left margin (default: 10pc)
  right:{spec}          right margin (default: 5pc)
  top:{spec}            top margin (default: 5pc)
  bottom:{spec}         bottom margin (default: 5pc)
                        {spec} is a number followed by "in" for inches, "pt"
                        for points (1 point is 1/72 of an inch), "mm" for
                        millimeters or "pc" for a percentage of the media
                        size.
                        Weird example:
                            left:2in,top:30pt,right:16mm,bottom:3pc
                        If the unit is not recognized there is no error and
                        the default value is used.

  header:{nr}           Number of lines to reserve for the header.
                        Only the first line is actually filled, thus when {nr}
                        is 2 there is one empty line.  The header is formatted
                        according to 'printheader'.
  header:0              Do not print a header.
  header:2  (default)   Use two lines for the header

  syntax:n              Do not use syntax highlighting.  This is faster and
                        thus useful when printing large files.
  syntax:y              Do syntax highlighting.
  syntax:a  (default)   Use syntax highlighting if the printer appears to be
                        able to print color or grey.

  number:y              Include line numbers in the printed output.
  number:n  (default)   No line numbers.

  wrap:y    (default)   Wrap long lines.
  wrap:n                Truncate long lines.

  duplex:off            Print on one side.
  duplex:long (default) Print on both sides (when possible), bind on long
                        side.
  duplex:short          Print on both sides (when possible), bind on short
                        side.
  collate:y  (default)  Collating: 1 2 3, 1 2 3, 1 2 3
  collate:n             No collating: 1 1 1, 2 2 2, 3 3 3

  jobsplit:n (default)  Do all copies in one print job
  jobsplit:y            Do each copy as a separate print job.  Useful when
                        doing N-up postprocessing.

  portrait:y (default)  Orientation is portrait.
  portrait:n            Orientation is landscape.
                                                *a4* *letter*
  paper:A4   (default)  Paper size: A4
  paper:{name}          Paper size from this table:
                        {name}      size in cm       size in inch ~
                        10x14       25.4  x 35.57    10    x 14
                        A3          29.7  x 42       11.69 x 16.54
                        A4          21    x 29.7      8.27 x 11.69
                        A5          14.8  x 21        5.83 x  8.27
                        B4          25    x 35.3     10.12 x 14.33
                        B5          17.6  x 25        7.17 x 10.12
                        executive   18.42 x 26.67     7.25 x 10.5
                        folio       21    x 33        8.27 x 13
                        ledger      43.13 x 27.96    17    x 11
                        legal       21.59 x 35.57     8.5  x 14
                        letter      21.59 x 27.96     8.5  x 11
                        quarto      21.59 x 27.5      8.5  x 10.83
                        statement   13.97 x 21.59     5.5  x  8.5
                        tabloid     27.96 x 43.13    11    x 17

  formfeed:n (default)  Treat form feed characters (0x0c) as a normal print
                        character.
  formfeed:y            When a form feed character is encountered, continue
                        printing of the current line at the beginning of the
                        first line on a new page.

The item indicated with (default) is used when the item is not present.  The
values are not always used, especially when using a dialog to select the
printer and options.
Example: >
        :set printoptions=paper:letter,duplex:off

==============================================================================
3. PostScript Printing                                  *postscript-printing*
                                                *E455* *E456* *E457* *E624*
Provided you have enough disk space there should be no problems generating a
PostScript file.  You need to have the runtime files correctly installed (if
you can find the help files, they probably are).

There are currently a number of limitations with PostScript printing:

- 'printfont' - The font name is ignored (the Courier family is always used -
  it should be available on all PostScript printers) but the font size is
  used.

- 'printoptions' - The duplex setting is used when generating PostScript
  output, but it is up to the printer to take notice of the setting.  If the
  printer does not support duplex printing then it should be silently ignored.
  Some printers, however, don't print at all.

- 8-bit support - While a number of 8-bit print character encodings are
  supported it is possible that some characters will not print.  Whether a
  character will print depends on the font in the printer knowing the
  character.  Missing characters will be replaced with an upside down question
  mark, or a space if that character is also not known by the font.  It may be
  possible to get all the characters in an encoding to print by installing a
  new version of the Courier font family.

- Multi-byte support - Currently Vim will try to convert multi-byte characters
  to the 8-bit encoding specified by 'printencoding' (or latin1 if it is
  empty).  Any characters that are not successfully converted are shown as
  unknown characters.  Printing will fail if Vim cannot convert the multi-byte
  to the 8-bit encoding.

==============================================================================
4. Custom 8-bit Print Character Encodings       *postscript-print-encoding*
                                                                *E618* *E619*
To use your own print character encoding when printing 8-bit character data
you need to define your own PostScript font encoding vector.  Details on how
to define a font encoding vector is beyond the scope of this help file, but
you can find details in the PostScript Language Reference Manual, 3rd Edition,
published by Addison-Wesley and available in PDF form at
http://www.adobe.com/.  The following describes what you need to do for Vim to
locate and use your print character encoding.

i.   Decide on a unique name for your encoding vector, one that does not clash
     with any of the recognized or standard encoding names that Vim uses (see
     |encoding-names| for a list), and that no one else is likely to use.
ii.  Copy $VIMRUNTIME/print/latin1.ps to the print subdirectory in your
     'runtimepath' and rename it with your unique name.
iii. Edit your renamed copy of latin1.ps, replacing all occurrences of latin1
     with your unique name (don't forget the line starting %%Title:), and
     modify the array of glyph names to define your new encoding vector.  The
     array must have exactly 256 entries or you will not be able to print!
iv.  Within Vim, set 'printencoding' to your unique encoding name and then
     print your file.  Vim will now use your custom print character encoding.

Vim will report an error with the resource file if you change the order or
content of the first 3 lines, other than the name of the encoding on the line
starting %%Title: or the version number on the line starting %%Version:.

[Technical explanation for those that know PostScript - Vim looks for a file
with the same name as the encoding it will use when printing.  The file
defines a new PostScript Encoding resource called /VIM-name, where name is the
print character encoding Vim will use.]

==============================================================================
5. PostScript CJK Printing                      *postscript-cjk-printing*
                                                        *E673* *E674* *E675*

Vim supports printing of Chinese, Japanese, and Korean files.  Setting up Vim
to correctly print CJK files requires setting up a few more options.

Each of these countries has many standard character sets and encodings which
require that both be specified when printing.  In addition, CJK fonts normally
do not have the concept of italic glyphs and use different weight or stroke
style to achieve emphasis when printing.  This in turn requires a different
approach to specifying fonts to use when printing.

The encoding and character set are specified with the 'printencoding' and
'printmbcharset' options.  If 'printencoding' is not specified then 'encoding'
is used as normal.  If 'printencoding' is specified then characters will be
translated to this encoding for printing.  You should ensure that the encoding
is compatible with the character set needed for the file contents or some
characters may not appear when printed.
```

```
The fonts to use for CJK printing are specified with 'printmbfont'.  This
option allows you to specify different fonts to use when printing characters
which are syntax highlighted with the font styles normal, italic, bold and
bold-italic.

No CJK fonts are supplied with Vim.  There are some free Korean, Japanese, and
Traditional Chinese fonts available at:

  http://examples.oreilly.com/cjkvinfo/adobe/samples/

You can find descriptions of the various fonts in the read me file at

  http://examples.oreilly.de/english_examples/cjkvinfo/adobe/00README

Please read your printer documentation on how to install new fonts.

CJK fonts can be large containing several thousand glyphs, and it is not
uncommon to find that they only contain a subset of a national standard.  It
is not unusual to find the fonts to not include characters for codes in the
ASCII code range.  If you find half-width Roman characters are not appearing
in your printout then you should configure Vim to use the Courier font the
half-width ASCII characters with 'printmbfont'.  If your font does not include
other characters then you will need to find another font that does.

Another issue with ASCII characters, is that the various national character
sets specify a couple of different glyphs in the ASCII code range.  If you
print ASCII text using the national character set you may see some unexpected
characters.  If you want true ASCII code printing then you need to configure
Vim to output ASCII characters for the ASCII code range with 'printmbfont'.

It is possible to define your own multi-byte character set although this
should not be attempted lightly.  A discussion on the process if beyond the
scope of these help files.  You can find details on CMap (character map) files
in the document 'Adobe CMap and CIDFont Files Specification, Version 1.0',
available from http://www.adobe.com as a PDF file.

==============================================================================
6. PostScript Printing Troubleshooting          *postscript-print-trouble*
                                                                      *E621*
Usually the only sign of a problem when printing with PostScript is that your
printout does not appear.  If you are lucky you may get a printed page that
tells you the PostScript operator that generated the error that prevented the
print job completing.

There are a number of possible causes as to why the printing may have failed:

- Wrong version of the prolog resource file.  The prolog resource file
  contains some PostScript that Vim needs to be able to print.  Each version
  of Vim needs one particular version.  Make sure you have correctly installed
  the runtime files, and don't have any old versions of a file called prolog
  in the print directory in your 'runtimepath' directory.

- Paper size.  Some PostScript printers will abort printing a file if they do
  not support the requested paper size.  By default Vim uses A4 paper.  Find
  out what size paper your printer normally uses and set the appropriate paper
  size with 'printoptions'.  If you cannot find the name of the paper used,
  measure a sheet and compare it with the table of supported paper sizes listed
  for 'printoptions', using the paper that is closest in both width AND height.
  Note: The dimensions of actual paper may vary slightly from the ones listed.
  If there is no paper listed close enough, then you may want to try psresize
  from PSUtils, discussed below.
```

```
- Two-sided printing (duplex).  Normally a PostScript printer that does not
  support two-sided printing will ignore any request to do it.  However, some
  printers may abort the job altogether.  Try printing with duplex turned off.
  Note: Duplex prints can be achieved manually using PS utils - see below.

- Collated printing.  As with Duplex printing, most PostScript printers that
  do not support collating printouts will ignore a request to do so.  Some may
  not.  Try printing with collation turned off.

- Syntax highlighting.  Some print management code may prevent the generated
  PostScript file from being printed on a black and white printer when syntax
  highlighting is turned on, even if solid black is the only color used.  Try
  printing with syntax highlighting turned off.

A safe printoptions setting to try is: >

        :set printoptions=paper:A4,duplex:off,collate:n,syntax:n

Replace "A4" with the paper size that best matches your printer paper.

==============================================================================
7. PostScript Utilities                                *postscript-print-util*

7.1 Ghostscript

Ghostscript is a PostScript and PDF interpreter that can be used to display
and print on non-PostScript printers PostScript and PDF files.  It can also
generate PDF files from PostScript.

Ghostscript will run on a wide variety of platforms.

There are three available versions:

- AFPL Ghostscript (formerly Aladdin Ghostscript) which is free for
  non-commercial use.  It can be obtained from:

    http://www.cs.wisc.edu/~ghost/

- GNU Ghostscript which is available under the GNU General Public License.  It
  can be obtained from:

    ftp://mirror.cs.wisc.edu/pub/mirrors/ghost/gnu/

- A commercial version for inclusion in commercial products.

Additional information on Ghostscript can also be found at:

  http://www.ghostscript.com/

Support for a number of non PostScript printers is provided in the
distribution as standard, but if you cannot find support for your printer
check the Ghostscript site for other printers not included by default.


7.2 Ghostscript Previewers.

The interface to Ghostscript is very primitive so a number of graphical front
ends have been created.  These allow easier PostScript file selection,
previewing at different zoom levels, and printing.  Check supplied
documentation for full details.
```

```
X11

- Ghostview.  Obtainable from:

    http://www.cs.wisc.edu/~ghost/gv/

- gv.  Derived from Ghostview.  Obtainable from:

    http://wwwthep.physik.uni-mainz.de/~plass/gv/

  Copies (possibly not the most recent) can be found at:

    http://www.cs.wisc.edu/~ghost/gv/

OpenVMS

- Is apparently supported in the main code now (untested).  See:

    http://wwwthep.physik.uni-mainz.de/~plass/gv/

Windows and OS/2

- GSview.  Obtainable from:

    http://www.cs.wisc.edu/~ghost/gsview/

DOS

- ps_view.  Obtainable from:

    ftp://ftp.pg.gda.pl/pub/TeX/support/ps_view/
    ftp://ftp.dante.de/tex-archive/support/ps_view/

Linux

- GSview.  Linux version of the popular Windows and OS/2 previewer.
  Obtainable from:

    http://www.cs.wisc.edu/~ghost/gsview/

- BMV.  Different from Ghostview and gv in that it doesn't use X but svgalib.
  Obtainable from:

    ftp://sunsite.unc.edu/pub/Linux/apps/graphics/viewers/svga/bmv-1.2.tgz


7.3 PSUtils

PSUtils is a collection of utility programs for manipulating PostScript
documents.  Binary distributions are available for many platforms, as well as
the full source.  PSUtils can be found at:

  http://knackered.org/angus/psutils

The utilities of interest include:

- psnup.     Convert PS files for N-up printing.
- psselect.  Select page range and order of printing.
- psresize.  Change the page size.
- psbook.    Reorder and lay out pages ready for making a book.

The output of one program can be used as the input to the next, allowing for
```

```
complex print document creation.


N-UP PRINTING

The psnup utility takes an existing PostScript file generated from Vim and
convert it to an n-up version.  The simplest way to create a 2-up printout is
to first create a PostScript file with: >

        :hardcopy > test.ps

Then on your command line execute: >

        psnup -n 2 test.ps final.ps

Note: You may get warnings from some Ghostscript previewers for files produced
by psnup - these may safely be ignored.

Finally print the file final.ps to your PostScript printer with your
platform's print command.  (You will need to delete the two PostScript files
afterwards yourself.)  'printexpr' could be modified to perform this extra
step before printing.


ALTERNATE DUPLEX PRINTING

It is possible to achieve a poor man's version of duplex printing using the PS
utility psselect.  This utility has options -e and -o for printing just the
even or odd pages of a PS file respectively.

First generate a PS file with the 'hardcopy' command, then generate new
files with all the odd and even numbered pages with: >

        psselect -o test.ps odd.ps
        psselect -e test.ps even.ps

Next print odd.ps with your platform's normal print command.  Then take the
print output, turn it over and place it back in the paper feeder.  Now print
even.ps with your platform's print command.  All the even pages should now
appear on the back of the odd pages.

There are a couple of points to bear in mind:

1. Position of the first page.  If the first page is on top of the printout
   when printing the odd pages then you need to reverse the order that the odd
   pages are printed.  This can be done with the -r option to psselect.  This
   will ensure page 2 is printed on the back of page 1.
   Note: it is better to reverse the odd numbered pages rather than the even
   numbered in case there are an odd number of pages in the original PS file.

2. Paper flipping.  When turning over the paper with the odd pages printed on
   them you may have to either flip them horizontally (along the long edge) or
   vertically (along the short edge), as well as possibly rotating them 180
   degrees.  All this depends on the printer - it will be more obvious for
   desktop ink jets than for small office laser printers where the paper path
   is hidden from view.


==============================================================================
8. Formfeed Characters                                  *printing-formfeed*

By default Vim does not do any special processing of |formfeed| control
```

```
characters.  Setting the 'printoptions' formfeed item will make Vim recognize
formfeed characters and continue printing the current line at the beginning
of the first line on a new page.  The use of formfeed characters provides
rudimentary print control but there are certain things to be aware of.

Vim will always start printing a line (including a line number if enabled)
containing a formfeed character, even if it is the first character on the
line.  This means if a line starting with a formfeed character is the first
line of a page then Vim will print a blank page.

Since the line number is printed at the start of printing the line containing
the formfeed character, the remainder of the line printed on the new page
will not have a line number printed for it (in the same way as the wrapped
lines of a long line when wrap in 'printoptions' is enabled).

If the formfeed character is the last character on a line, then printing will
continue on the second line of the new page, not the first.  This is due to
Vim processing the end of the line after the formfeed character and moving
down a line to continue printing.

Due to the points made above it is recommended that when formfeed character
processing is enabled, printing of line numbers is disabled, and that form
feed characters are not the last character on a line.  Even then you may need
to adjust the number of lines before a formfeed character to prevent
accidental blank pages.

==============================================================================
 vim:tw=78:ts=8:ft=help:norl:
*remote.txt*    For Vim version 8.0.  Last change: 2017 Aug 01


                  VIM REFERENCE MANUAL    by Bram Moolenaar


Vim client-server communication                         *client-server*

1. Common functionality         |clientserver|
2. X11 specific items           |x11-clientserver|
3. MS-Windows specific items    |w32-clientserver|

{Vi does not have any of these commands}

==============================================================================
1. Common functionality                                 *clientserver*

When compiled with the |+clientserver| option, Vim can act as a command
server.  It accepts messages from a client and executes them.  At the same
time, Vim can function as a client and send commands to a Vim server.

The following command line arguments are available:

    argument                    meaning ~

   --remote [+{cmd}] {file} ...                                 *--remote*
                                Open the file list in a remote Vim.  When
                                there is no Vim server, execute locally.
                                There is one optional init command: +{cmd}.
                                This must be an Ex command that can be
                                followed by "|".
                                The rest of the command line is taken as the
                                file list.  Thus any non-file arguments must
                                come before this.
                                You cannot edit stdin this way |--|.
                                The remote Vim is raised.  If you don't want
                                this use >
                                 vim --remote-send "<C-\><C-N>:n filename<CR>"
<
   --remote-silent [+{cmd}] {file} ...                  *--remote-silent*
                                As above, but don't complain if there is no
                                server and the file is edited locally.
   --remote-wait [+{cmd}] {file} ...                            *--remote-wait*
                                As --remote, but wait for files to complete
                                (unload) in remote Vim.
   --remote-wait-silent [+{cmd}] {file} ...             *--remote-wait-silent*
                                As --remote-wait, but don't complain if there
                                is no server.
                                                        *--remote-tab*
   --remote-tab                 Like --remote but open each file in a new
                                tabpage.
                                                        *--remote-tab-silent*
   --remote-tab-silent          Like --remote-silent but open each file in a
                                new tabpage.
                                                        *--remote-tab-wait*
   --remote-tab-wait            Like --remote-wait but open each file in a new
                                tabpage.

                                                *--remote-tab-wait-silent*
   --remote-tab-wait-silent     Like --remote-wait-silent but open each file
                                in a new tabpage.
                                                                *--servername*
   --servername {name}          Become the server {name}.  When used together
                                with one of the --remote commands: connect to
                                server {name} instead of the default (see
                                below).
                                                                *--remote-send*
   --remote-send {keys}         Send {keys} to server and exit.  The {keys}
                                are not mapped.  Special key names are
                                recognized, e.g., "<CR>" results in a CR
                                character.
                                                                *--remote-expr*
   --remote-expr {expr}         Evaluate {expr} in server and print the result
                                on stdout.
                                                                *--serverlist*
   --serverlist                 Output a list of server names.


Examples ~

Edit "file.txt" in an already running GVIM server: >
    gvim --remote file.txt

Edit "file.txt" in an already running server called FOOBAR: >
    gvim --servername FOOBAR --remote file.txt

Edit "file.txt" in server "FILES" if it exists, become server "FILES"
otherwise: >
    gvim --servername FILES --remote-silent file.txt

This doesn't work, all arguments after --remote will be used as file names: >
    gvim --remote --servername FOOBAR file.txt

Edit file "+foo" in a remote server (note the use of "./" to avoid the special
meaning of the leading plus): >
    vim --remote ./+foo
```

```
Tell the remote server "BLA" to write all files and exit: >
    vim --servername BLA --remote-send '<C-\><C-N>:wqa<CR>'


SERVER NAME                                             *client-server-name*

By default Vim will try to register the name under which it was invoked (gvim,
egvim ...).  This can be overridden with the --servername argument.  If the
specified name is not available, a postfix is applied until a free name is
encountered, i.e. "gvim1" for the second invocation of gvim on a particular
X-server.  The resulting name is available in the servername builtin variable
|v:servername|.  The case of the server name is ignored, thus "gvim" and
"GVIM" are considered equal.

When Vim is invoked with --remote, --remote-wait or --remote-send it will try
to locate the server name determined by the invocation name and --servername
argument as described above.  If an exact match is not available, the first
server with the number postfix will be used.  If a name with the number
postfix is specified with the --servername argument, it must match exactly.

If no server can be located and --remote or --remote-wait was used, Vim will
start up according to the rest of the command line and do the editing by
itself.  This way it is not necessary to know whether gvim is already started
when sending command to it.

The --serverlist argument will cause Vim to print a list of registered command
servers on the standard output (stdout) and exit.

Win32 Note: Making the Vim server go to the foreground doesn't always work,
because MS-Windows doesn't allow it.  The client will move the server to the
foreground when using the --remote or --remote-wait argument and the server
name starts with "g".


REMOTE EDITING

The --remote argument will cause a |:drop| command to be constructed from the
rest of the command line and sent as described above.
The --remote-wait argument does the same thing and additionally sets up to
wait for each of the files to have been edited.  This uses the BufUnload
event, thus as soon as a file has been unloaded, Vim assumes you are done
editing it.
Note that the --remote and --remote-wait arguments will consume the rest of
the command line.  I.e. all remaining arguments will be regarded as filenames.
You can not put options there!


FUNCTIONS
                                                                *E240* *E573*
There are a number of Vim functions for scripting the command server.  See
the description in |eval.txt| or use CTRL-] on the function name to jump to
the full explanation.

    synopsis                                explanation ~
    remote_startserver( name)               run a server
    remote_expr( server, string, idvar)     send expression
    remote_send( server, string, idvar)     send key sequence
    serverlist()                            get a list of available servers
    remote_peek( serverid, retvar)          check for reply string
    remote_read( serverid)                  read reply string
    server2client( serverid, string)        send reply string
    remote_foreground( server)              bring server to the front
```

See also the explanation of |CTRL-_CTRL-N|. Very useful as a leading key
sequence.
The {serverid} for server2client() can be obtained with expand("<client>")

```
==============================================================================
2. X11 specific items                                   *x11-clientserver*
                                    *E247* *E248* *E251* *E258* *E277*
```

The communication between client and server goes through the X server. The
display of the Vim server must be specified. The usual protection of the X
server is used, you must be able to open a window on the X server for the
communication to work. It is possible to communicate between different
systems.

By default, a GUI Vim will register a name on the X-server by which it can be
addressed for subsequent execution of injected strings. Vim can also act as
a client and send strings to other instances of Vim on the same X11 display.

When an X11 GUI Vim (gvim) is started, it will try to register a send-server
name on the 'VimRegistry' property on the root window.

A non GUI Vim with access to the X11 display (|xterm-clipboard| enabled), can
also act as a command server if a server name is explicitly given with the
--servername argument.

An empty --servername argument will cause the command server to be disabled.

To send commands to a Vim server from another application, read the source
file src/if_xcmdsrv.c, it contains some hints about the protocol used.

```
==============================================================================
3. Win32 specific items                                 *w32-clientserver*
```

Every Win32 Vim can work as a server, also in the console. You do not need a
version compiled with OLE. Windows messages are used, this works on any
version of MS-Windows. But only communication within one system is possible.

Since MS-Windows messages are used, any other application should be able to
communicate with a Vim server. An alternative is using the OLE functionality
|ole-interface|.

When using gvim, the --remote-wait only works properly this way: >

```
        start /w gvim --remote-wait file.txt
<
 vim:tw=78:sw=4:ts=8:ft=help:norl:
*term.txt*      For Vim version 8.0.  Last change: 2017 Aug 28


                  VIM REFERENCE MANUAL    by Bram Moolenaar


Terminal information                                    *terminal-info*
```

Vim uses information about the terminal you are using to fill the screen and
recognize what keys you hit. If this information is not correct, the screen
may be messed up or keys may not be recognized. The actions which have to be
performed on the screen are accomplished by outputting a string of
characters. Special keys produce a string of characters. These strings are
stored in the terminal options, see |terminal-options|.

NOTE: Most of this is not used when running the |GUI|.

==

1. Startup   *startup-terminal*

When Vim is started a default terminal type is assumed. For the Amiga this is a standard CLI window, for MS-DOS the pc terminal, for Unix an ansi terminal. A few other terminal types are always available, see below |builtin-terms|.

You can give the terminal name with the '-T' Vim argument. If it is not given Vim will try to get the name from the TERM environment variable.

termcap *terminfo* *E557* *E558* *E559*

On Unix the terminfo database or termcap file is used. This is referred to as "termcap" in all the documentation. At compile time, when running configure, the choice whether to use terminfo or termcap is done automatically. When running Vim the output of ":version" will show |+terminfo| if terminfo is used. Also see |xterm-screens|.

On non-Unix systems a termcap is only available if Vim was compiled with TERMCAP defined.

builtin-terms *builtin_terms*

Which builtin terminals are available depends on a few defines in feature.h, which need to be set at compile time:

define	output of ":version"	terminals builtin ~
NO_BUILTIN_TCAPS	-builtin_terms	none
SOME_BUILTIN_TCAPS	+builtin_terms	most common ones (default)
ALL_BUILTIN_TCAPS	++builtin_terms	all available

You can see a list of available builtin terminals with ":set term=xxx" (when not running the GUI). Also see |+builtin_terms|.

If the termcap code is included Vim will try to get the strings for the terminal you are using from the termcap file and the builtin termcaps. Both are always used, if an entry for the terminal you are using is present. Which one is used first depends on the 'ttybuiltin' option:

'ttybuiltin' on	1: builtin termcap	2: external termcap
'ttybuiltin' off	1: external termcap	2: builtin termcap

If an option is missing in one of them, it will be obtained from the other one. If an option is present in both, the one first encountered is used.

Which external termcap file is used varies from system to system and may depend on the environment variables "TERMCAP" and "TERMPATH". See "man tgetent".

Settings depending on terminal   *term-dependent-settings*

If you want to set options or mappings, depending on the terminal name, you can do this best in your .vimrc. Example: >

```
   if &term == "xterm"
     ... xterm maps and settings ...
   elseif &term =~ "vt10."
     ... vt100, vt102 maps and settings ...
   endif
<
                                                *raw-terminal-mode*
For normal editing the terminal will be put into "raw" mode.  The strings
defined with 't_ti' and 't_ks' will be sent to the terminal.  Normally this
puts the terminal in a state where the termcap codes are valid and activates
the cursor and function keys.  When Vim exits the terminal will be put back
into the mode it was before Vim started.  The strings defined with 't_te' and
't_ke' will be sent to the terminal.  On the Amiga, with commands that execute
an external command (e.g., "!!"), the terminal will be put into Normal mode
for a moment.  This means that you can stop the output to the screen by
hitting a printing key.  Output resumes when you hit <BS>.

                                                *xterm-bracketed-paste*
When the 't_BE' option is set then 't_BE' will be sent to the
terminal when entering "raw" mode and 't_BD' when leaving "raw" mode.  The
terminal is then expected to put 't_PS' before pasted text and 't_PE' after
pasted text.  This way Vim can separate text that is pasted from characters
that are typed.  The pasted text is handled like when the middle mouse button
is used, it is inserted literally and not interpreted as commands.

When the cursor is in the first column, the pasted text will be inserted
before it.  Otherwise the pasted text is appended after the cursor position.
This means one cannot paste after the first column.  Unfortunately Vim does
not have a way to tell where the mouse pointer was.

Note that in some situations Vim will not recognize the bracketed paste and
you will get the raw text.  In other situations Vim will only get the first
pasted character and drop the rest, e.g. when using the "r" command.  If you
have a problem with this, disable bracketed paste by putting this in your
.vimrc: >
        set t_BE=
If this is done while Vim is running the 't_BD' will be sent to the terminal
to disable bracketed paste.

                                                        *cs7-problem*
Note: If the terminal settings are changed after running Vim, you might have
an illegal combination of settings.  This has been reported on Solaris 2.5
with "stty cs8 parenb", which is restored as "stty cs7 parenb".  Use
"stty cs8 -parenb -istrip" instead, this is restored correctly.

Some termcap entries are wrong in the sense that after sending 't_ks' the
cursor keys send codes different from the codes defined in the termcap.  To
avoid this you can set 't_ks' (and 't_ke') to empty strings.  This must be
done during initialization (see |initialization|), otherwise it's too late.

Some termcap entries assume that the highest bit is always reset.  For
example: The cursor-up entry for the Amiga could be ":ku=\E[A:".  But the
Amiga really sends "\233A".  This works fine if the highest bit is reset,
e.g., when using an Amiga over a serial line.  If the cursor keys don't work,
try the entry ":ku=\233A:".

Some termcap entries have the entry ":ku=\E[A:".  But the Amiga really sends
"\233A".  On output "\E[" and "\233" are often equivalent, on input they
aren't.  You will have to change the termcap entry, or change the key code with
the :set command to fix this.

Many cursor key codes start with an <Esc>.  Vim must find out if this is a
single hit of the <Esc> key or the start of a cursor key sequence.  It waits
for a next character to arrive.  If it does not arrive within one second a
```

```
single <Esc> is assumed.  On very slow systems this may fail, causing cursor
keys not to work sometimes.  If you discover this problem reset the 'timeout'
option.  Vim will wait for the next character to arrive after an <Esc>.  If
you want to enter a single <Esc> you must type it twice.  Resetting the
'esckeys' option avoids this problem in Insert mode, but you lose the
possibility to use cursor and function keys in Insert mode.

On the Amiga the recognition of window resizing is activated only when the
terminal name is "amiga" or "builtin_amiga".

Some terminals have confusing codes for the cursor keys.  The televideo 925 is
such a terminal.  It sends a CTRL-H for cursor-left.  This would make it
impossible to distinguish a backspace and cursor-left.  To avoid this problem
CTRL-H is never recognized as cursor-left.

                                        *vt100-cursor-keys* *xterm-cursor-keys*
Other terminals (e.g., vt100 and xterm) have cursor keys that send <Esc>OA,
<Esc>OB, etc.  Unfortunately these are valid commands in insert mode: Stop
insert, Open a new line above the new one, start inserting 'A', 'B', etc.
Instead of performing these commands Vim will erroneously recognize this typed
key sequence as a cursor key movement.  To avoid this and make Vim do what you
want in either case you could use these settings: >
        :set notimeout          " don't timeout on mappings
        :set ttimeout           " do timeout on terminal key codes
        :set timeoutlen=100     " timeout after 100 msec
This requires the key-codes to be sent within 100 msec in order to recognize
them as a cursor key.  When you type you normally are not that fast, so they
are recognized as individual typed commands, even though Vim receives the same
sequence of bytes.

                                *vt100-function-keys* *xterm-function-keys*
An xterm can send function keys F1 to F4 in two modes: vt100 compatible or
not.  Because Vim may not know what the xterm is sending, both types of keys
are recognized.  The same happens for the <Home> and <End> keys.
                        normal                  vt100 ~
        <F1>    t_k1    <Esc>[11~       <xF1>   <Esc>OP     *<xF1>-xterm*
        <F2>    t_k2    <Esc>[12~       <xF2>   <Esc>OQ     *<xF2>-xterm*
        <F3>    t_k3    <Esc>[13~       <xF3>   <Esc>OR     *<xF3>-xterm*
        <F4>    t_k4    <Esc>[14~       <xF4>   <Esc>OS     *<xF4>-xterm*
        <Home>  t_kh    <Esc>[7~        <xHome> <Esc>OH     *<xHome>-xterm*
        <End>   t_@7    <Esc>[4~        <xEnd>  <Esc>OF     *<xEnd>-xterm*

When Vim starts, <xF1> is mapped to <F1>, <xF2> to <F2> etc.  This means that
by default both codes do the same thing.  If you make a mapping for <xF2>,
because your terminal does have two keys, the default mapping is overwritten,
thus you can use the <F2> and <xF2> keys for something different.

                                                        *xterm-shifted-keys*
Newer versions of xterm support shifted function keys and special keys.  Vim
recognizes most of them.  Use ":set termcap" to check which are supported and
what the codes are.  Mostly these are not in a termcap, they are only
supported by the builtin_xterm termcap.

                                                        *xterm-modifier-keys*
Newer versions of xterm support Alt and Ctrl for most function keys.  To avoid
having to add all combinations of Alt, Ctrl and Shift for every key a special
sequence is recognized at the end of a termcap entry: ";*X".  The "X" can be
any character, often '~' is used.  The ";*" stands for an optional modifier
argument.  ";2" is Shift, ";3" is Alt, ";5" is Ctrl and ";9" is Meta (when
it's different from Alt).  They can be combined.  Examples: >
        :set <F8>=^[[19;*~
        :set <Home>=^[[1;*H
```

Another speciality about these codes is that they are not overwritten by
another code. That is to avoid that the codes obtained from xterm directly
|t_RV| overwrite them.

xterm-scroll-region

The default termcap entry for xterm on Sun and other platforms does not
contain the entry for scroll regions. Add ":cs=\E[%i%d;%dr:" to the xterm
entry in /etc/termcap and everything should work.

xterm-end-home-keys

On some systems (at least on FreeBSD with XFree86 3.1.2) the codes that the
<End> and <Home> keys send contain a <Nul> character. To make these keys send
the proper key code, add these lines to your ~/.Xdefaults file:

```
*VT100.Translations:            #override \n\
                <Key>Home: string("0x1b") string("[7~") \n\
                <Key>End: string("0x1b") string("[8~")
```

xterm-8bit *xterm-8-bit*

Xterm can be run in a mode where it uses 8-bit escape sequences. The CSI code
is used instead of <Esc>[. The advantage is that an <Esc> can quickly be
recognized in Insert mode, because it can't be confused with the start of a
special key.
For the builtin termcap entries, Vim checks if the 'term' option contains
"8bit" anywhere. It then uses 8-bit characters for the termcap entries, the
mouse and a few other things. You would normally set $TERM in your shell to
"xterm-8bit" and Vim picks this up and adjusts to the 8-bit setting
automatically.
When Vim receives a response to the |t_RV| (request version) sequence and it
starts with CSI, it assumes that the terminal is in 8-bit mode and will
convert all key sequences to their 8-bit variants.

==

2. Terminal options *terminal-options* *termcap-options* *E436*

The terminal options can be set just like normal options. But they are not
shown with the ":set all" command. Instead use ":set termcap".

It is always possible to change individual strings by setting the
appropriate option. For example: >

```
        :set t_ce=^V^[[K        (CTRL-V, <Esc>, [, K)
```

{Vi: no terminal options. You have to exit Vi, edit the termcap entry and
try again}

The options are listed below. The associated termcap code is always equal to
the last two characters of the option name. Only one termcap code is
required: Cursor motion, 't_cm'.

The options 't_da', 't_db', 't_ms', 't_xs', 't_xn' represent flags in the
termcap. When the termcap flag is present, the option will be set to "y".
But any non-empty string means that the flag is set. An empty string means
that the flag is not set. 't_CS' works like this too, but it isn't a termcap
flag.

OUTPUT CODES *terminal-output-codes*

option	meaning ~	
t_AB	set background color (ANSI)	*t_AB* *'t_AB'*
t_AF	set foreground color (ANSI)	*t_AF* *'t_AF'*
t_AL	add number of blank lines	*t_AL* *'t_AL'*
t_al	add new blank line	*t_al* *'t_al'*
t_bc	backspace character	*t_bc* *'t_bc'*

```
        t_cd    clear to end of screen                          *t_cd* *'t_cd'*
        t_ce    clear to end of line                            *t_ce* *'t_ce'*
        t_cl    clear screen                                    *t_cl* *'t_cl'*
        t_cm    cursor motion (required!)                 *E437* *t_cm* *'t_cm'*
        t_Co    number of colors                                *t_Co* *'t_Co'*
        t_CS    if non-empty, cursor relative to scroll region  *t_CS* *'t_CS'*
        t_cs    define scrolling region                         *t_cs* *'t_cs'*
        t_CV    define vertical scrolling region                *t_CV* *'t_CV'*
        t_da    if non-empty, lines from above scroll down      *t_da* *'t_da'*
        t_db    if non-empty, lines from below scroll up        *t_db* *'t_db'*
        t_DL    delete number of lines                          *t_DL* *'t_DL'*
        t_dl    delete line                                     *t_dl* *'t_dl'*
        t_fs    set window title end (from status line)         *t_fs* *'t_fs'*
        t_ke    exit "keypad transmit" mode                     *t_ke* *'t_ke'*
        t_ks    start "keypad transmit" mode                    *t_ks* *'t_ks'*
        t_le    move cursor one char left                       *t_le* *'t_le'*
        t_mb    blinking mode                                   *t_mb* *'t_mb'*
        t_md    bold mode                                       *t_md* *'t_md'*
        t_me    Normal mode (undoes t_mr, t_mb, t_md and color) *t_me* *'t_me'*
        t_mr    reverse (invert) mode                           *t_mr* *'t_mr'*
                                                                *t_ms* *'t_ms'*
        t_ms    if non-empty, cursor can be moved in standout/inverse mode
        t_nd    non destructive space character                 *t_nd* *'t_nd'*
        t_op    reset to original color pair                    *t_op* *'t_op'*
        t_RI    cursor number of chars right                    *t_RI* *'t_RI'*
        t_Sb    set background color                            *t_Sb* *'t_Sb'*
        t_Sf    set foreground color                            *t_Sf* *'t_Sf'*
        t_se    standout end                                    *t_se* *'t_se'*
        t_so    standout mode                                   *t_so* *'t_so'*
        t_sr    scroll reverse (backward)                       *t_sr* *'t_sr'*
        t_te    out of "termcap" mode                           *t_te* *'t_te'*
        t_ti    put terminal in "termcap" mode                  *t_ti* *'t_ti'*
        t_ts    set window title start (to status line)         *t_ts* *'t_ts'*
        t_ue    underline end                                   *t_ue* *'t_ue'*
        t_us    underline mode                                  *t_us* *'t_us'*
        t_ut    clearing uses the current background color      *t_ut* *'t_ut'*
        t_vb    visual bell                                     *t_vb* *'t_vb'*
        t_ve    cursor visible                                  *t_ve* *'t_ve'*
        t_vi    cursor invisible                                *t_vi* *'t_vi'*
        t_vs    cursor very visible (blink)                     *t_vs* *'t_vs'*
                                                                *t_xs* *'t_xs'*
        t_xs    if non-empty, standout not erased by overwriting (hpterm)
                                                                *t_xn* *'t_xn'*
        t_xn    if non-empty, writing a character at the last screen cell
                does not cause scrolling
        t_ZH    italics mode                                    *t_ZH* *'t_ZH'*
        t_ZR    italics end                                     *t_ZR* *'t_ZR'*

Added by Vim (there are no standard codes for these):
        t_Ce    undercurl end                                   *t_Ce* *'t_Ce'*
        t_Cs    undercurl mode                                  *t_Cs* *'t_Cs'*
        t_Te    strikethrough end                               *t_Te* *'t_Te'*
        t_Ts    strikethrough mode                              *t_Ts* *'t_Ts'*
        t_IS    set icon text start                             *t_IS* *'t_IS'*
        t_IE    set icon text end                               *t_IE* *'t_IE'*
        t_WP    set window position (Y, X) in pixels            *t_WP* *'t_WP'*
        t_GP    get window position (Y, X) in pixels            *t_GP* *'t_GP'*
        t_WS    set window size (height, width in cells)        *t_WS* *'t_WS'*
        t_VS    cursor normally visible (no blink)              *t_VS* *'t_VS'*
        t_SI    start insert mode (bar cursor shape)            *t_SI* *'t_SI'*
        t_SR    start replace mode (underline cursor shape)     *t_SR* *'t_SR'*
        t_EI    end insert or replace mode (block cursor shape) *t_EI* *'t_EI'*
```

```
                |termcap-cursor-shape|
        t_RV    request terminal version string (for xterm)     *t_RV* *'t_RV'*
                |xterm-8bit| |v:termresponse| |'ttymouse'| |xterm-codes|
        t_u7    request cursor position (for xterm)             *t_u7* *'t_u7'*
                see |'ambiwidth'|
        t_RB    request terminal background color               *t_RB* *'t_RB'*
        t_8f    set foreground color (R, G, B)                  *t_8f* *'t_8f'*
                |xterm-true-color|
        t_8b    set background color (R, G, B)                  *t_8b* *'t_8b'*
                |xterm-true-color|
        t_BE    enable bracketed paste mode                     *t_BE* *'t_BE'*
                |xterm-bracketed-paste|
        t_BD    disable bracketed paste mode                    *t_BD* *'t_BD'*
                |xterm-bracketed-paste|
        t_SC    set cursor color start                          *t_SC* *'t_SC'*
        t_EC    set cursor color end                            *t_EC* *'t_EC'*
        t_SH    set cursor shape                                *t_SH* *'t_SH'*
        t_RC    request terminal cursor blinking                *t_RC* *'t_RC'*
        t_RS    request terminal cursor style                   *t_RS* *'t_RS'*

Some codes have a start, middle and end part.  The start and end are defined
by the termcap option, the middle part is text.
        set title text:     t_ts {title text} t_fs
        set icon text:      t_IS {icon text} t_IE
        set cursor color:   t_SC  {color name}  t_EC

t_SH must take one argument:
        0, 1 or none    blinking block cursor
        2               block cursor
        3               blinking underline cursor
        4               underline cursor
        5               blinking vertical bar cursor
        6               vertical bar cursor

t_RS is sent only if the response to t_RV has been received.  It is not used
on Mac OS when Terminal.app could be recognized from the termresponse.


KEY CODES                                               *terminal-key-codes*
Note: Use the <> form if possible

        option  name            meaning ~

        t_ku    <Up>            arrow up                        *t_ku* *'t_ku'*
        t_kd    <Down>          arrow down                      *t_kd* *'t_kd'*
        t_kr    <Right>         arrow right                     *t_kr* *'t_kr'*
        t_kl    <Left>          arrow left                      *t_kl* *'t_kl'*
                <xUp>           alternate arrow up              *<xUp>*
                <xDown>         alternate arrow down            *<xDown>*
                <xRight>        alternate arrow right           *<xRight>*
                <xLeft>         alternate arrow left            *<xLeft>*
                <S-Up>          shift arrow up
                <S-Down>        shift arrow down
        t_%i    <S-Right>       shift arrow right               *t_%i* *'t_%i'*
        t_#4    <S-Left>        shift arrow left                *t_#4* *'t_#4'*
        t_k1    <F1>            function key 1                  *t_k1* *'t_k1'*
                <xF1>           alternate F1                    *<xF1>*
        t_k2    <F2>            function key 2          *<F2>*  *t_k2* *'t_k2'*
                <xF2>           alternate F2                    *<xF2>*
        t_k3    <F3>            function key 3          *<F3>*  *t_k3* *'t_k3'*
                <xF3>           alternate F3                    *<xF3>*
        t_k4    <F4>            function key 4          *<F4>*  *t_k4* *'t_k4'*
                <xF4>           alternate F4                    *<xF4>*
        t_k5    <F5>            function key 5          *<F5>*  *t_k5* *'t_k5'*
        t_k6    <F6>            function key 6          *<F6>*  *t_k6* *'t_k6'*
        t_k7    <F7>            function key 7          *<F7>*  *t_k7* *'t_k7'*
        t_k8    <F8>            function key 8          *<F8>*  *t_k8* *'t_k8'*
        t_k9    <F9>            function key 9          *<F9>*  *t_k9* *'t_k9'*
        t_k;    <F10>           function key 10         *<F10>* *t_k;* *'t_k;'*
        t_F1    <F11>           function key 11         *<F11>* *t_F1* *'t_F1'*
        t_F2    <F12>           function key 12         *<F12>* *t_F2* *'t_F2'*
        t_F3    <F13>           function key 13         *<F13>* *t_F3* *'t_F3'*
        t_F4    <F14>           function key 14         *<F14>* *t_F4* *'t_F4'*
        t_F5    <F15>           function key 15         *<F15>* *t_F5* *'t_F5'*
        t_F6    <F16>           function key 16         *<F16>* *t_F6* *'t_F6'*
        t_F7    <F17>           function key 17         *<F17>* *t_F7* *'t_F7'*
        t_F8    <F18>           function key 18         *<F18>* *t_F8* *'t_F8'*
        t_F9    <F19>           function key 19         *<F19>* *t_F9* *'t_F9'*
                <S-F1>          shifted function key 1
                <S-xF1>         alternate <S-F1>                *<S-xF1>*
                <S-F2>          shifted function key 2          *<S-F2>*
                <S-xF2>         alternate <S-F2>                *<S-xF2>*
                <S-F3>          shifted function key 3          *<S-F3>*
                <S-xF3>         alternate <S-F3>                *<S-xF3>*
                <S-F4>          shifted function key 4          *<S-F4>*
                <S-xF4>         alternate <S-F4>                *<S-xF4>*
                <S-F5>          shifted function key 5          *<S-F5>*
                <S-F6>          shifted function key 6          *<S-F6>*
                <S-F7>          shifted function key 7          *<S-F7>*
                <S-F8>          shifted function key 8          *<S-F8>*
                <S-F9>          shifted function key 9          *<S-F9>*
                <S-F10>         shifted function key 10         *<S-F10>*
                <S-F11>         shifted function key 11         *<S-F11>*
                <S-F12>         shifted function key 12         *<S-F12>*
        t_%1    <Help>          help key                        *t_%1* *'t_%1'*
        t_&8    <Undo>          undo key                        *t_&8* *'t_&8'*
        t_kI    <Insert>        insert key                      *t_kI* *'t_kI'*
        t_kD    <Del>           delete key                      *t_kD* *'t_kD'*
        t_kb    <BS>            backspace key                   *t_kb* *'t_kb'*
        t_kB    <S-Tab>         back-tab (shift-tab)  *<S-Tab>* *t_kB* *'t_kB'*
        t_kh    <Home>          home key                        *t_kh* *'t_kh'*
        t_#2    <S-Home>        shifted home key     *<S-Home>* *t_#2* *'t_#2'*
                <xHome>         alternate home key              *<xHome>*
        t_@7    <End>           end key                         *t_@7* *'t_@7'*
        t_*7    <S-End>         shifted end key *<S-End>* *t_star7* *'t_star7'*
                <xEnd>          alternate end key               *<xEnd>*
        t_kP    <PageUp>        page-up key                     *t_kP* *'t_kP'*
        t_kN    <PageDown>      page-down key                   *t_kN* *'t_kN'*
        t_K1    <kHome>         keypad home key                 *t_K1* *'t_K1'*
        t_K4    <kEnd>          keypad end key                  *t_K4* *'t_K4'*
        t_K3    <kPageUp>       keypad page-up key              *t_K3* *'t_K3'*
        t_K5    <kPageDown>     keypad page-down key            *t_K5* *'t_K5'*
        t_K6    <kPlus>         keypad plus key       *<kPlus>* *t_K6* *'t_K6'*
        t_K7    <kMinus>        keypad minus key     *<kMinus>* *t_K7* *'t_K7'*
        t_K8    <kDivide>       keypad divide       *<kDivide>* *t_K8* *'t_K8'*
        t_K9    <kMultiply>     keypad multiply   *<kMultiply>* *t_K9* *'t_K9'*
        t_KA    <kEnter>        keypad enter key     *<kEnter>* *t_KA* *'t_KA'*
        t_KB    <kPoint>        keypad decimal point *<kPoint>* *t_KB* *'t_KB'*
        t_KC    <k0>            keypad 0                 *<k0>* *t_KC* *'t_KC'*
        t_KD    <k1>            keypad 1                 *<k1>* *t_KD* *'t_KD'*
        t_KE    <k2>            keypad 2                 *<k2>* *t_KE* *'t_KE'*
        t_KF    <k3>            keypad 3                 *<k3>* *t_KF* *'t_KF'*
        t_KG    <k4>            keypad 4                 *<k4>* *t_KG* *'t_KG'*
        t_KH    <k5>            keypad 5                 *<k5>* *t_KH* *'t_KH'*
```

```
        t_KI    <k6>            keypad 6                *<k6>* *t_KI* *'t_KI'*
        t_KJ    <k7>            keypad 7                *<k7>* *t_KJ* *'t_KJ'*
        t_KK    <k8>            keypad 8                *<k8>* *t_KK* *'t_KK'*
        t_KL    <k9>            keypad 9                *<k9>* *t_KL* *'t_KL'*
                <Mouse>         leader of mouse code           *<Mouse>*
                                                               *t_PS* *'t_PS'*
        t_PS    start of bracketed paste |xterm-bracketed-paste|
        t_PE    end of bracketed paste |xterm-bracketed-paste|  *t_PE* *'t_PE'*
```

Note about t_so and t_mr: When the termcap entry "so" is not present the
entry for "mr" is used. And vice versa. The same is done for "se" and "me".
If your terminal supports both inversion and standout mode, you can see two
different modes. If your terminal supports only one of the modes, both will
look the same.

```
                                                        *keypad-comma*
```
The keypad keys, when they are not mapped, behave like the equivalent normal
key. There is one exception: if you have a comma on the keypad instead of a
decimal point, Vim will use a dot anyway. Use these mappings to fix that: >
```
        :noremap <kPoint> ,
        :noremap! <kPoint> ,
<                                                       *xterm-codes*
```
There is a special trick to obtain the key codes which currently only works
for xterm. When |t_RV| is defined and a response is received which indicates
an xterm with patchlevel 141 or higher, Vim uses special escape sequences to
request the key codes directly from the xterm. The responses are used to
adjust the various t_ codes. This avoids the problem that the xterm can
produce different codes, depending on the mode it is in (8-bit, VT102,
VT220, etc.). The result is that codes like <xF1> are no longer needed.
Note: This is only done on startup. If the xterm options are changed after
Vim has started, the escape sequences may not be recognized anymore.

```
                                                        *xterm-true-color*
```
Vim supports using true colors in the terminal (taken from |highlight-guifg|
and |highlight-guibg|), given that the terminal supports this. To make this
work the 'termguicolors' option needs to be set.
See https://gist.github.com/XVilka/8346728 for a list of terminals that
support true colors.

Sometimes setting 'termguicolors' is not enough and one has to set the |t_8f|
and |t_8b| options explicitly. Default values of these options are
"^[[38;2;%lu;%lu;%lum" and "^[[48;2;%lu;%lu;%lum" respectively, but it is only
set when `$TERM` is `xterm`. Some terminals accept the same sequences, but
with all semicolons replaced by colons (this is actually more compatible, but
less widely supported): >
```
         let &t_8f = "\<Esc>[38:2:%lu:%lu:%lum"
         let &t_8b = "\<Esc>[48:2:%lu:%lu:%lum"
```

These options contain printf strings, with |printf()| (actually, its C
equivalent hence `l` modifier) invoked with the t_ option value and three
unsigned long integers that may have any value between 0 and 255 (inclusive)
representing red, green and blue colors respectively.

```
                                                        *xterm-resize*
```
Window resizing with xterm only works if the allowWindowOps resource is
enabled. On some systems and versions of xterm it's disabled by default
because someone thought it would be a security issue. It's not clear if this
is actually the case.

To overrule the default, put this line in your ~/.Xdefaults or
~/.Xresources:
>

```
        XTerm*allowWindowOps:           true

And run "xrdb -merge .Xresources" to make it effective.  You can check the
value with the context menu (right mouse button while CTRL key is pressed),
there should be a tick at allow-window-ops.

                                                        *termcap-colors*
Note about colors: The 't_Co' option tells Vim the number of colors available.
When it is non-zero, the 't_AB' and 't_AF' options are used to set the color.
If one of these is not available, 't_Sb' and 't_Sf' are used.  't_me' is used
to reset to the default colors.

                                *termcap-cursor-shape* *termcap-cursor-color*
When Vim enters Insert mode the 't_SI' escape sequence is sent.  When Vim
enters Replace mode the 't_SR' escape sequence is sent if it is set, otherwise
't_SI' is sent.  When leaving Insert mode or Replace mode 't_EI' is used. This
can be used to change the shape or color of the cursor in Insert or Replace
mode. These are not standard termcap/terminfo entries, you need to set them
yourself.
Example for an xterm, this changes the color of the cursor: >
    if &term =~ "xterm"
        let &t_SI = "\<Esc>]12;purple\x7"
        let &t_SR = "\<Esc>]12;red\x7"
        let &t_EI = "\<Esc>]12;blue\x7"
    endif
NOTE: When Vim exits the shape for Normal mode will remain.  The shape from
before Vim started will not be restored.
{not available when compiled without the |+cursorshape| feature}

                                                        *termcap-title*
The 't_ts' and 't_fs' options are used to set the window title if the terminal
allows title setting via sending strings.  They are sent before and after the
title string, respectively.  Similar 't_IS' and 't_IE'  are used to set the
icon text.  These are Vim-internal extensions of the Unix termcap, so they
cannot be obtained from an external termcap.  However, the builtin termcap
contains suitable entries for xterm and iris-ansi, so you don't need to set
them here.
                                                        *hpterm*
If inversion or other highlighting does not work correctly, try setting the
't_xs' option to a non-empty string.  This makes the 't_ce' code be used to
remove highlighting from a line.  This is required for "hpterm".  Setting the
'weirdinvert' option has the same effect as making 't_xs' non-empty, and vice
versa.

                                                        *scroll-region*
Some termcaps do not include an entry for 'cs' (scroll region), although the
terminal does support it.  For example: xterm on a Sun.  You can use the
builtin_xterm or define t_cs yourself.  For example: >
        :set t_cs=^V^[[%i%d;%dr
Where ^V is CTRL-V and ^[ is <Esc>.

The vertical scroll region t_CV is not a standard termcap code.  Vim uses it
internally in the GUI.  But it can also be defined for a terminal, if you can
find one that supports it.  The two arguments are the left and right column of
the region which to restrict the scrolling to.  Just like t_cs defines the top
and bottom lines.  Defining t_CV will make scrolling in vertically split
windows a lot faster.  Don't set t_CV when t_da or t_db is set (text isn't
cleared when scrolling).

Unfortunately it is not possible to deduce from the termcap how cursor
positioning should be done when using a scrolling region: Relative to the
beginning of the screen or relative to the beginning of the scrolling region.
```

Most terminals use the first method. A known exception is the MS-DOS console (pcterm). The 't_CS' option should be set to any string when cursor positioning is relative to the start of the scrolling region. It should be set to an empty string otherwise. It defaults to "yes" when 'term' is "pcterm".

```
Note for xterm users: The shifted cursor keys normally don't work.  You can
        make them work with the xmodmap command and some mappings in Vim.

        Give these commands in the xterm:
                xmodmap -e "keysym Up = Up F13"
                xmodmap -e "keysym Down = Down F16"
                xmodmap -e "keysym Left = Left F18"
                xmodmap -e "keysym Right = Right F19"

        And use these mappings in Vim:
                :map <t_F3> <S-Up>
                :map! <t_F3> <S-Up>
                :map <t_F6> <S-Down>
                :map! <t_F6> <S-Down>
                :map <t_F8> <S-Left>
                :map! <t_F8> <S-Left>
                :map <t_F9> <S-Right>
                :map! <t_F9> <S-Right>
```

Instead of, say, <S-Up> you can use any other command that you want to use the shift-cursor-up key for. (Note: To help people that have a Sun keyboard with left side keys F14 is not used because it is confused with the undo key; F15 is not used, because it does a window-to-front; F17 is not used, because it closes the window. On other systems you can probably use them.)

==
3. Window size   *window-size*

[This is about the size of the whole window Vim is using, not a window that is created with the ":split" command.]

If you are running Vim on an Amiga and the terminal name is "amiga" or "builtin_amiga", the amiga-specific window resizing will be enabled. On Unix systems three methods are tried to get the window size:

- an ioctl call (TIOCGSIZE or TIOCGWINSZ, depends on your system)
- the environment variables "LINES" and "COLUMNS"
- from the termcap entries "li" and "co"

If everything fails a default size of 24 lines and 80 columns is assumed. If a window-resize signal is received the size will be set again. If the window size is wrong you can use the 'lines' and 'columns' options to set the correct values.

One command can be used to set the screen size:

```
                                                *:mod* *:mode* *E359*
:mod[e] [mode]
```

Without argument this only detects the screen size and redraws the screen. With MS-DOS it is possible to switch screen mode. [mode] can be one of these values:

```
        "bw40"          40 columns black&white
        "c40"           40 columns color
        "bw80"          80 columns black&white
        "c80"           80 columns color (most people use this)
        "mono"          80 columns monochrome
        "c4350"         43 or 50 lines EGA/VGA mode
        number          mode number to use, depends on your video card

==============================================================================
4. Slow and fast terminals                      *slow-fast-terminal*
                                                *slow-terminal*

If you have a fast terminal you may like to set the 'ruler' option.  The
cursor position is shown in the status line.  If you are using horizontal
scrolling ('wrap' option off) consider setting 'sidescroll' to a small
number.

If you have a slow terminal you may want to reset the 'showcmd' option.
The command characters will not be shown in the status line.  If the terminal
scrolls very slowly, set the 'scrolljump' to 5 or so.  If the cursor is moved
off the screen (e.g., with "j") Vim will scroll 5 lines at a time.  Another
possibility is to reduce the number of lines that Vim uses with the command
"z{height}<CR>".

If the characters from the terminal are arriving with more than 1 second
between them you might want to set the 'timeout' and/or 'ttimeout' option.
See the "Options" chapter |options|.

If your terminal does not support a scrolling region, but it does support
insert/delete line commands, scrolling with multiple windows may make the
lines jump up and down.  If you don't want this set the 'ttyfast' option.
This will redraw the window instead of scroll it.

If your terminal scrolls very slowly, but redrawing is not slow, set the
'ttyscroll' option to a small number, e.g., 3.  This will make Vim redraw the
screen instead of scrolling, when there are more than 3 lines to be scrolled.

If you are using a color terminal that is slow, use this command: >
        hi NonText cterm=NONE ctermfg=NONE
This avoids that spaces are sent when they have different attributes.  On most
terminals you can't see this anyway.

If you are using Vim over a slow serial line, you might want to try running
Vim inside the "screen" program.  Screen will optimize the terminal I/O quite
a bit.

If you are testing termcap options, but you cannot see what is happening,
you might want to set the 'writedelay' option.  When non-zero, one character
is sent to the terminal at a time (does not work for MS-DOS).  This makes the
screen updating a lot slower, making it possible to see what is happening.

==============================================================================
5. Using the mouse                                      *mouse-using*

This section is about using the mouse on a terminal or a terminal window.  How
to use the mouse in a GUI window is explained in |gui-mouse|.  For scrolling
with a mouse wheel see |scroll-mouse-wheel|.

Don't forget to enable the mouse with this command: >
        :set mouse=a
Otherwise Vim won't recognize the mouse in all modes (See 'mouse').

Currently the mouse is supported for Unix in an xterm window, in a *BSD
console with |sysmouse|, in a Linux console (with GPM |gpm-mouse|), for
MS-DOS and in a Windows console.
Mouse clicks can be used to position the cursor, select an area and paste.
```

These characters in the 'mouse' option tell in which situations the mouse will be used by Vim:

n	Normal mode
v	Visual mode
i	Insert mode
c	Command-line mode
h	all previous modes when in a help file
a	all previous modes
r	for |hit-enter| prompt

The default for 'mouse' is empty, the mouse is not used. Normally you would do: >

```
	:set mouse=a
```

to start using the mouse (this is equivalent to setting 'mouse' to "nvich"). If you only want to use the mouse in a few modes or also want to use it for the two questions you will have to concatenate the letters for those modes. For example: >

```
	:set mouse=nv
```

Will make the mouse work in Normal mode and Visual mode. >

```
	:set mouse=h
```

Will make the mouse work in help files only (so you can use "g<LeftMouse>" to jump to tags).

Whether the selection that is started with the mouse is in Visual mode or Select mode depends on whether "mouse" is included in the 'selectmode' option.

In an xterm, with the currently active mode included in the 'mouse' option, normal mouse clicks are used by Vim, mouse clicks with the shift or ctrl key pressed go to the xterm. With the currently active mode not included in 'mouse' all mouse clicks go to the xterm.

xterm-clipboard

In the Athena and Motif GUI versions, when running in a terminal and there is access to the X-server (DISPLAY is set), the copy and paste will behave like in the GUI. If not, the middle mouse button will insert the unnamed register. In that case, here is how you copy and paste a piece of text:

Copy/paste with the mouse and Visual mode ('mouse' option must be set, see above):
1. Press left mouse button on first letter of text, move mouse pointer to last letter of the text and release the button. This will start Visual mode and highlight the selected area.
2. Press "y" to yank the Visual text in the unnamed register.
3. Click the left mouse button at the insert position.
4. Click the middle mouse button.

Shortcut: If the insert position is on the screen at the same time as the Visual text, you can do 2, 3 and 4 all in one: Click the middle mouse button at the insert position.

Note: When the |-X| command line argument is used, Vim will not connect to the X server and copy/paste to the X clipboard (selection) will not work. Use the shift key with the mouse buttons to let the xterm do the selection.

xterm-command-server

When the X-server clipboard is available, the command server described in |x11-clientserver| can be enabled with the --servername command line argument.

xterm-copy-paste

NOTE: In some (older) xterms, it's not possible to move the cursor past column

95 or 223. This is an xterm problem, not Vim's. Get a newer xterm |color-xterm|. Also see |'ttymouse'|.

Copy/paste in xterm with (current mode NOT included in 'mouse'):
1. Press left mouse button on first letter of text, move mouse pointer to last
   letter of the text and release the button.
2. Use normal Vim commands to put the cursor at the insert position.
3. Press "a" to start Insert mode.
4. Click the middle mouse button.
5. Press ESC to end Insert mode.

(The same can be done with anything in 'mouse' if you keep the shift key pressed while using the mouse.)

Note: if you lose the 8th bit when pasting (special characters are translated into other characters), you may have to do "stty cs8 -istrip -parenb" in your shell before starting Vim.

Thus in an xterm the shift and ctrl keys cannot be used with the mouse. Mouse commands requiring the CTRL modifier can be simulated by typing the "g" key before using the mouse:

```
        "g<LeftMouse>"  is "<C-LeftMouse>       (jump to tag under mouse click)
        "g<RightMouse>" is "<C-RightMouse>      ("CTRL-T")
```

mouse-mode-table *mouse-overview*

A short overview of what the mouse buttons do, when 'mousemodel' is "extend":

Normal Mode:

event	position	selection	change	action	~
	cursor		window		~
<LeftMouse>	yes	end	yes		
<C-LeftMouse>	yes	end	yes	"CTRL-]" (2)	
<S-LeftMouse>	yes	no change	yes	"*" (2)	*<S-LeftMouse>*
<LeftDrag>	yes	start or extend (1)	no		*<LeftDrag>*
<LeftRelease>	yes	start or extend (1)	no		
<MiddleMouse>	yes	if not active	no	put	
<MiddleMouse>	yes	if active	no	yank and put	
<RightMouse>	yes	start or extend	yes		
<A-RightMouse>	yes	start or extend blockw.	yes		*<A-RightMouse>*
<S-RightMouse>	yes	no change	yes	"#" (2)	*<S-RightMouse>*
<C-RightMouse>	no	no change	no	"CTRL-T"	
<RightDrag>	yes	extend	no		*<RightDrag>*
<RightRelease>	yes	extend	no		*<RightRelease>*

Insert or Replace Mode:

event	position	selection	change	action	~
	cursor		window		~
<LeftMouse>	yes	(cannot be active)	yes		
<C-LeftMouse>	yes	(cannot be active)	yes	"CTRL-O^]" (2)	
<S-LeftMouse>	yes	(cannot be active)	yes	"CTRL-O*" (2)	
<LeftDrag>	yes	start or extend (1)	no	like CTRL-O (1)	
<LeftRelease>	yes	start or extend (1)	no	like CTRL-O (1)	
<MiddleMouse>	no	(cannot be active)	no	put register	
<RightMouse>	yes	start or extend	yes	like CTRL-O	
<A-RightMouse>	yes	start or extend blockw.	yes		
<S-RightMouse>	yes	(cannot be active)	yes	"CTRL-O#" (2)	
<C-RightMouse>	no	(cannot be active)	no	"CTRL-O CTRL-T"	

In a help window:

event	position	selection	change	action	~
	cursor		window		~
<2-LeftMouse>	yes	(cannot be active)	no	"^]" (jump to help tag)	

```
When 'mousemodel' is "popup", these are different:

Normal Mode:
event         position     selection      change  action        ~
               cursor                     window                ~
<S-LeftMouse>   yes     start or extend (1) no
<A-LeftMouse>   yes start or extend blockw. no                *<A-LeftMouse>*
<RightMouse>    no      popup menu          no

Insert or Replace Mode:
event         position     selection      change  action        ~
               cursor                     window                ~
<S-LeftMouse>   yes     start or extend (1) no     like CTRL-O (1)
<A-LeftMouse>   yes start or extend blockw. no
<RightMouse>    no      popup menu          no

(1) only if mouse pointer moved since press
(2) only if click is in same buffer

Clicking the left mouse button causes the cursor to be positioned.  If the
click is in another window that window is made the active window.  When
editing the command-line the cursor can only be positioned on the
command-line.  When in Insert mode Vim remains in Insert mode.  If 'scrolloff'
is set, and the cursor is positioned within 'scrolloff' lines from the window
border, the text is scrolled.

A selection can be started by pressing the left mouse button on the first
character, moving the mouse to the last character, then releasing the mouse
button.  You will not always see the selection until you release the button,
only in some versions (GUI, MS-DOS, WIN32) will the dragging be shown
immediately.  Note that you can make the text scroll by moving the mouse at
least one character in the first/last line in the window when 'scrolloff' is
non-zero.

In Normal, Visual and Select mode clicking the right mouse button causes the
Visual area to be extended.  When 'mousemodel' is "popup", the left button has
to be used while keeping the shift key pressed.  When clicking in a window
which is editing another buffer, the Visual or Select mode is stopped.

In Normal, Visual and Select mode clicking the right mouse button with the alt
key pressed causes the Visual area to become blockwise.  When 'mousemodel' is
"popup" the left button has to be used with the alt key.  Note that this won't
work on systems where the window manager consumes the mouse events when the
alt key is pressed (it may move the window).

                                                        *double-click*
Double, triple and quadruple clicks are supported when the GUI is active,
for MS-DOS and Win32, and for an xterm (if the gettimeofday() function is
available).  For selecting text, extra clicks extend the selection:
        click           select ~
        double          word or % match         *<2-LeftMouse>*
        triple          line                    *<3-LeftMouse>*
        quadruple       rectangular block       *<4-LeftMouse>*
Exception: In a Help window a double click jumps to help for the word that is
clicked on.
A double click on a word selects that word.  'iskeyword' is used to specify
which characters are included in a word.  A double click on a character
that has a match selects until that match (like using "v%").  If the match is
an #if/#else/#endif block, the selection becomes linewise.
For MS-DOS and xterm the time for double clicking can be set with the
'mousetime' option.  For the other systems this time is defined outside of
Vim.
```

An example, for using a double click to jump to the tag under the cursor: >

```
        :map <2-LeftMouse> :exe "tag ". expand("<cword>")<CR>
```

Dragging the mouse with a double click (button-down, button-up, button-down and then drag) will result in whole words to be selected. This continues until the button is released, at which point the selection is per character again.

gpm-mouse

The GPM mouse is only supported when the |+mouse_gpm| feature was enabled at compile time. The GPM mouse driver (Linux console) does not support quadruple clicks.

In Insert mode, when a selection is started, Vim goes into Normal mode temporarily. When Visual or Select mode ends, it returns to Insert mode. This is like using CTRL-O in Insert mode. Select mode is used when the 'selectmode' option contains "mouse".

sysmouse

The sysmouse is only supported when the |+mouse_sysmouse| feature was enabled at compile time. The sysmouse driver (*BSD console) does not support keyboard modifiers.

drag-status-line

When working with several windows, the size of the windows can be changed by dragging the status line with the mouse. Point the mouse at a status line, press the left button, move the mouse to the new position of the status line, release the button. Just clicking the mouse in a status line makes that window the current window, without moving the cursor. If by selecting a window it will change position or size, the dragging of the status line will look confusing, but it will work (just try it).

<MiddleRelease> *<MiddleDrag>*

Mouse clicks can be mapped. The codes for mouse clicks are:

code	mouse button	normal action	~
<LeftMouse>	left pressed	set cursor position	
<LeftDrag>	left moved while pressed	extend selection	
<LeftRelease>	left released	set selection end	
<MiddleMouse>	middle pressed	paste text at cursor position	
<MiddleDrag>	middle moved while pressed	-	
<MiddleRelease>	middle released	-	
<RightMouse>	right pressed	extend selection	
<RightDrag>	right moved while pressed	extend selection	
<RightRelease>	right released	set selection end	
<X1Mouse>	X1 button pressed	-	*X1Mouse*
<X1Drag>	X1 moved while pressed	-	*X1Drag*
<X1Release>	X1 button release	-	*X1Release*
<X2Mouse>	X2 button pressed	-	*X2Mouse*
<X2Drag>	X2 moved while pressed	-	*X2Drag*
<X2Release>	X2 button release	-	*X2Release*

The X1 and X2 buttons refer to the extra buttons found on some mice. The 'Microsoft Explorer' mouse has these buttons available to the right thumb. Currently X1 and X2 only work on Win32 and X11 environments.

Examples: >

```
        :noremap <MiddleMouse> <LeftMouse><MiddleMouse>
```

Paste at the position of the middle mouse button click (otherwise the paste would be done at the cursor position). >

```
        :noremap <LeftRelease> <LeftRelease>y
```

Immediately yank the selection, when using Visual mode.

```
Note the use of ":noremap" instead of "map" to avoid a recursive mapping.
>
	:map <X1Mouse> <C-O>
	:map <X2Mouse> <C-I>
Map the X1 and X2 buttons to go forwards and backwards in the jump list, see
|CTRL-O| and |CTRL-I|.

						*mouse-swap-buttons*
To swap the meaning of the left and right mouse buttons: >
	:noremap	<LeftMouse>	<RightMouse>
	:noremap	<LeftDrag>	<RightDrag>
	:noremap	<LeftRelease>	<RightRelease>
	:noremap	<RightMouse>	<LeftMouse>
	:noremap	<RightDrag>	<LeftDrag>
	:noremap	<RightRelease>	<LeftRelease>
	:noremap	g<LeftMouse>	<C-RightMouse>
	:noremap	g<RightMouse>	<C-LeftMouse>
	:noremap!	<LeftMouse>	<RightMouse>
	:noremap!	<LeftDrag>	<RightDrag>
	:noremap!	<LeftRelease>	<RightRelease>
	:noremap!	<RightMouse>	<LeftMouse>
	:noremap!	<RightDrag>	<LeftDrag>
	:noremap!	<RightRelease>	<LeftRelease>
<
 vim:tw=78:ts=8:ft=help:norl:
*digraph.txt*   For Vim version 8.0.  Last change: 2016 Nov 04


		  VIM REFERENCE MANUAL    by Bram Moolenaar


Digraphs					*digraph* *digraphs* *Digraphs*

Digraphs are used to enter characters that normally cannot be entered by
an ordinary keyboard.  These are mostly printable non-ASCII characters.  The
digraphs are easier to remember than the decimal number that can be entered
with CTRL-V (see |i_CTRL-V|).

There is a brief introduction on digraphs in the user manual: |24.9|
An alternative is using the 'keymap' option.

1. Defining digraphs	|digraphs-define|
2. Using digraphs	|digraphs-use|
3. Default digraphs	|digraphs-default|

{Vi does not have any of these commands}

==============================================================================
1. Defining digraphs					*digraphs-define*

						*:dig* *:digraphs*
:dig[raphs]		show currently defined digraphs.
							*E104* *E39*
:dig[raphs] {char1}{char2} {number} ...
			Add digraph {char1}{char2} to the list.  {number} is
			the decimal representation of the character.  Normally
			it is the Unicode character, see |digraph-encoding|.
			Example: >
	:digr e: 235 a: 228
<			Avoid defining a digraph with '_' (underscore) as the
			first character, it has a special meaning in the
			future.
```

Vim is normally compiled with the |+digraphs| feature. If the feature is
disabled, the ":digraph" command will display an error message.

Example of the output of ":digraphs": >

```
 TH Þ  222  ss ß  223  a! à  224  a' á  225  a> â  226  a? ã  227  a: ä  228
```

The first two characters in each column are the characters you have to type to
enter the digraph.

In the middle of each column is the resulting character. This may be mangled
if you look at it on a system that does not support digraphs or if you print
this file.

digraph-encoding

The decimal number normally is the Unicode number of the character. Note that
the meaning doesn't change when 'encoding' changes. The character will be
converted from Unicode to 'encoding' when needed. This does require the
conversion to be available, it might fail. For the NUL character you will see
"10". That's because NUL characters are internally represented with a NL
character. When you write the file it will become a NUL character.

When Vim was compiled without the |+multi_byte| feature, you need to specify
the character in the encoding given with 'encoding'. You might want to use
something like this: >

```
        if has("multi_byte")
                digraph oe 339
        elseif &encoding == "iso-8859-15"
                digraph oe 189
        endif
```

This defines the "oe" digraph for a character that is number 339 in Unicode
and 189 in latin9 (iso-8859-15).

==

2. Using digraphs   *digraphs-use*

There are two methods to enter digraphs: *i_digraph*

```
        CTRL-K {char1} {char2}          or
        {char1} <BS> {char2}
```

The first is always available; the second only when the 'digraph' option is
set.

If a digraph with {char1}{char2} does not exist, Vim searches for a digraph
{char2}{char1}. This helps when you don't remember which character comes
first.

Note that when you enter CTRL-K {char1}, where {char1} is a special key, Vim
enters the code for that special key. This is not a digraph.

Once you have entered the digraph, Vim treats the character like a normal
character that occupies only one character in the file and on the screen.
Example: >

```
        'B' <BS> 'B'    will enter the broken '|' character (166)
        'a' <BS> '>'    will enter an 'a' with a circumflex (226)
        CTRL-K '-' '-'  will enter a soft hyphen (173)
```

The current digraphs are listed with the ":digraphs" command. Some of the
default ones are listed below |digraph-table|.

For CTRL-K, there is one general digraph: CTRL-K <Space> {char} will enter {char} with the highest bit set. You can use this to enter meta-characters.

The <Esc> character cannot be part of a digraph. When hitting <Esc>, Vim stops digraph entry and ends Insert mode or Command-line mode, just like hitting an <Esc> out of digraph context. Use CTRL-V 155 to enter meta-ESC (CSI).

If you accidentally typed an 'a' that should be an 'e', you will type 'a' <BS> 'e'. But that is a digraph, so you will not get what you want. To correct this, you will have to type <BS> e again. To avoid this don't set the 'digraph' option and use CTRL-K to enter digraphs.

You may have problems using Vim with characters which have a value above 128. For example: You insert ue (u-umlaut) and the editor echoes \334 in Insert mode. After leaving the Insert mode everything is fine. Note that fmt removes all characters with a value above 128 from the text being formatted. On some Unix systems this means you have to define the environment-variable LC_CTYPE. If you are using csh, then put the following line in your .cshrc: >

```
	setenv LC_CTYPE iso_8859_1
```

==

3. Default digraphs *digraphs-default*

Vim comes with a set of default digraphs. Check the output of ":digraphs" to see them.

On most systems Vim uses the same digraphs. They work for the Unicode and ISO-8859-1 character sets. These default digraphs are taken from the RFC1345 mnemonics. To make it easy to remember the mnemonic, the second character has a standard meaning:

char name	char	meaning ~
Exclamation mark	!	Grave
Apostrophe	'	Acute accent
Greater-Than sign	>	Circumflex accent
Question mark	?	Tilde
Hyphen-Minus	-	Macron
Left parenthesis	(	Breve
Full stop	.	Dot above
Colon	:	Diaeresis
Comma	,	Cedilla
Underline	_	Underline
Solidus	/	Stroke
Quotation mark	"	Double acute accent
Semicolon	;	Ogonek
Less-Than sign	<	Caron
Zero	0	Ring above
Two	2	Hook
Nine	9	Horn
Equals	=	Cyrillic (= used as second char)
Asterisk	*	Greek
Percent sign	%	Greek/Cyrillic special
Plus	+	smalls: Arabic, capitals: Hebrew
Three	3	some Latin/Greek/Cyrillic letters
Four	4	Bopomofo
Five	5	Hiragana
Six	6	Katakana

Example: a: is ä and o: is ö

These are the RFC1345 digraphs for the one-byte characters. See the output of

":digraphs" for the others. The characters above 255 are only available when
Vim was compiled with the |+multi_byte| feature.

EURO

Exception: RFC1345 doesn't specify the euro sign. In Vim the digraph =e was
added for this. Note the difference between latin1, where the digraph Cu is
used for the currency sign, and latin9 (iso-8859-15), where the digraph =e is
used for the euro sign, while both of them are the character 164, 0xa4. For
compatibility with zsh Eu can also be used for the euro sign.

ROUBLE

The rouble sign was added in 2014 as 0x20bd. Vim supports the digraphs =R and
=P for this. Note that R= and P= are other characters.

digraph-table

char	digraph	hex	dec	official name ~
^@	NU	0x00	0	NULL (NUL)
^A	SH	0x01	1	START OF HEADING (SOH)
^B	SX	0x02	2	START OF TEXT (STX)
^C	EX	0x03	3	END OF TEXT (ETX)
^D	ET	0x04	4	END OF TRANSMISSION (EOT)
^E	EQ	0x05	5	ENQUIRY (ENQ)
^F	AK	0x06	6	ACKNOWLEDGE (ACK)
^G	BL	0x07	7	BELL (BEL)
^H	BS	0x08	8	BACKSPACE (BS)
^I	HT	0x09	9	CHARACTER TABULATION (HT)
^@	LF	0x0a	10	LINE FEED (LF)
^K	VT	0x0b	11	LINE TABULATION (VT)
^L	FF	0x0c	12	FORM FEED (FF)
^M	CR	0x0d	13	CARRIAGE RETURN (CR)
^N	SO	0x0e	14	SHIFT OUT (SO)
^O	SI	0x0f	15	SHIFT IN (SI)
^P	DL	0x10	16	DATALINK ESCAPE (DLE)
^Q	D1	0x11	17	DEVICE CONTROL ONE (DC1)
^R	D2	0x12	18	DEVICE CONTROL TWO (DC2)
^S	D3	0x13	19	DEVICE CONTROL THREE (DC3)
^T	D4	0x14	20	DEVICE CONTROL FOUR (DC4)
^U	NK	0x15	21	NEGATIVE ACKNOWLEDGE (NAK)
^V	SY	0x16	22	SYNCHRONOUS IDLE (SYN)
^W	EB	0x17	23	END OF TRANSMISSION BLOCK (ETB)
^X	CN	0x18	24	CANCEL (CAN)
^Y	EM	0x19	25	END OF MEDIUM (EM)
^Z	SB	0x1a	26	SUBSTITUTE (SUB)
^[	EC	0x1b	27	ESCAPE (ESC)
^\	FS	0x1c	28	FILE SEPARATOR (IS4)
^]	GS	0x1d	29	GROUP SEPARATOR (IS3)
^^	RS	0x1e	30	RECORD SEPARATOR (IS2)
^_	US	0x1f	31	UNIT SEPARATOR (IS1)
	SP	0x20	32	SPACE
#	Nb	0x23	35	NUMBER SIGN
$	DO	0x24	36	DOLLAR SIGN
@	At	0x40	64	COMMERCIAL AT
[	<(	0x5b	91	LEFT SQUARE BRACKET
\	//	0x5c	92	REVERSE SOLIDUS
]	)>	0x5d	93	RIGHT SQUARE BRACKET
^	'>	0x5e	94	CIRCUMFLEX ACCENT
`	'!	0x60	96	GRAVE ACCENT
{	(!	0x7b	123	LEFT CURLY BRACKET
|	!!	0x7c	124	VERTICAL LINE
}	!)	0x7d	125	RIGHT CURLY BRACKET

```
~       '?      0x7e    126     TILDE
^?      DT      0x7f    127     DELETE (DEL)
~@      PA      0x80    128     PADDING CHARACTER (PAD)
~A      HO      0x81    129     HIGH OCTET PRESET (HOP)
~B      BH      0x82    130     BREAK PERMITTED HERE (BPH)
~C      NH      0x83    131     NO BREAK HERE (NBH)
~D      IN      0x84    132     INDEX (IND)
~E      NL      0x85    133     NEXT LINE (NEL)
~F      SA      0x86    134     START OF SELECTED AREA (SSA)
~G      ES      0x87    135     END OF SELECTED AREA (ESA)
~H      HS      0x88    136     CHARACTER TABULATION SET (HTS)
~I      HJ      0x89    137     CHARACTER TABULATION WITH JUSTIFICATION (HTJ)
~J      VS      0x8a    138     LINE TABULATION SET (VTS)
~K      PD      0x8b    139     PARTIAL LINE FORWARD (PLD)
~L      PU      0x8c    140     PARTIAL LINE BACKWARD (PLU)
~M      RI      0x8d    141     REVERSE LINE FEED (RI)
~N      S2      0x8e    142     SINGLE-SHIFT TWO (SS2)
~O      S3      0x8f    143     SINGLE-SHIFT THREE (SS3)
~P      DC      0x90    144     DEVICE CONTROL STRING (DCS)
~Q      P1      0x91    145     PRIVATE USE ONE (PU1)
~R      P2      0x92    146     PRIVATE USE TWO (PU2)
~S      TS      0x93    147     SET TRANSMIT STATE (STS)
~T      CC      0x94    148     CANCEL CHARACTER (CCH)
~U      MW      0x95    149     MESSAGE WAITING (MW)
~V      SG      0x96    150     START OF GUARDED AREA (SPA)
~W      EG      0x97    151     END OF GUARDED AREA (EPA)
~X      SS      0x98    152     START OF STRING (SOS)
~Y      GC      0x99    153     SINGLE GRAPHIC CHARACTER INTRODUCER (SGCI)
~Z      SC      0x9a    154     SINGLE CHARACTER INTRODUCER (SCI)
~[      CI      0x9b    155     CONTROL SEQUENCE INTRODUCER (CSI)
~\      ST      0x9c    156     STRING TERMINATOR (ST)
~]      OC      0x9d    157     OPERATING SYSTEM COMMAND (OSC)
~^      PM      0x9e    158     PRIVACY MESSAGE (PM)
~_      AC      0x9f    159     APPLICATION PROGRAM COMMAND (APC)
|       NS      0xa0    160     NO-BREAK SPACE
¡       !I      0xa1    161     INVERTED EXCLAMATION MARK
¢       Ct      0xa2    162     CENT SIGN
£       Pd      0xa3    163     POUND SIGN
¤       Cu      0xa4    164     CURRENCY SIGN
¥       Ye      0xa5    165     YEN SIGN
¦       BB      0xa6    166     BROKEN BAR
§       SE      0xa7    167     SECTION SIGN
¨       ':      0xa8    168     DIAERESIS
©       Co      0xa9    169     COPYRIGHT SIGN
ª       -a      0xaa    170     FEMININE ORDINAL INDICATOR
«       <<      0xab    171     LEFT-POINTING DOUBLE ANGLE QUOTATION MARK
¬       NO      0xac    172     NOT SIGN
        --      0xad    173     SOFT HYPHEN
®       Rg      0xae    174     REGISTERED SIGN
¯       'm      0xaf    175     MACRON
°       DG      0xb0    176     DEGREE SIGN
±       +-      0xb1    177     PLUS-MINUS SIGN
²       2S      0xb2    178     SUPERSCRIPT TWO
³       3S      0xb3    179     SUPERSCRIPT THREE
´       ''      0xb4    180     ACUTE ACCENT
µ       My      0xb5    181     MICRO SIGN
¶       PI      0xb6    182     PILCROW SIGN
·       .M      0xb7    183     MIDDLE DOT
¸       ',      0xb8    184     CEDILLA
¹       1S      0xb9    185     SUPERSCRIPT ONE
º       -o      0xba    186     MASCULINE ORDINAL INDICATOR
»       >>      0xbb    187     RIGHT-POINTING DOUBLE ANGLE QUOTATION MARK
```

¼	14	0xbc	188	VULGAR FRACTION ONE QUARTER
½	12	0xbd	189	VULGAR FRACTION ONE HALF
¾	34	0xbe	190	VULGAR FRACTION THREE QUARTERS
¿	?I	0xbf	191	INVERTED QUESTION MARK
À	A!	0xc0	192	LATIN CAPITAL LETTER A WITH GRAVE
Á	A'	0xc1	193	LATIN CAPITAL LETTER A WITH ACUTE
Â	A>	0xc2	194	LATIN CAPITAL LETTER A WITH CIRCUMFLEX
Ã	A?	0xc3	195	LATIN CAPITAL LETTER A WITH TILDE
Ä	A:	0xc4	196	LATIN CAPITAL LETTER A WITH DIAERESIS
Å	AA	0xc5	197	LATIN CAPITAL LETTER A WITH RING ABOVE
Æ	AE	0xc6	198	LATIN CAPITAL LETTER AE
Ç	C,	0xc7	199	LATIN CAPITAL LETTER C WITH CEDILLA
È	E!	0xc8	200	LATIN CAPITAL LETTER E WITH GRAVE
É	E'	0xc9	201	LATIN CAPITAL LETTER E WITH ACUTE
Ê	E>	0xca	202	LATIN CAPITAL LETTER E WITH CIRCUMFLEX
Ë	E:	0xcb	203	LATIN CAPITAL LETTER E WITH DIAERESIS
Ì	I!	0xcc	204	LATIN CAPITAL LETTER I WITH GRAVE
Í	I'	0xcd	205	LATIN CAPITAL LETTER I WITH ACUTE
Î	I>	0xce	206	LATIN CAPITAL LETTER I WITH CIRCUMFLEX
Ï	I:	0xcf	207	LATIN CAPITAL LETTER I WITH DIAERESIS
Ð	D-	0xd0	208	LATIN CAPITAL LETTER ETH (Icelandic)
Ñ	N?	0xd1	209	LATIN CAPITAL LETTER N WITH TILDE
Ò	O!	0xd2	210	LATIN CAPITAL LETTER O WITH GRAVE
Ó	O'	0xd3	211	LATIN CAPITAL LETTER O WITH ACUTE
Ô	O>	0xd4	212	LATIN CAPITAL LETTER O WITH CIRCUMFLEX
Õ	O?	0xd5	213	LATIN CAPITAL LETTER O WITH TILDE
Ö	O:	0xd6	214	LATIN CAPITAL LETTER O WITH DIAERESIS
×	*X	0xd7	215	MULTIPLICATION SIGN
Ø	O/	0xd8	216	LATIN CAPITAL LETTER O WITH STROKE
Ù	U!	0xd9	217	LATIN CAPITAL LETTER U WITH GRAVE
Ú	U'	0xda	218	LATIN CAPITAL LETTER U WITH ACUTE
Û	U>	0xdb	219	LATIN CAPITAL LETTER U WITH CIRCUMFLEX
Ü	U:	0xdc	220	LATIN CAPITAL LETTER U WITH DIAERESIS
Ý	Y'	0xdd	221	LATIN CAPITAL LETTER Y WITH ACUTE
Þ	TH	0xde	222	LATIN CAPITAL LETTER THORN (Icelandic)
ß	ss	0xdf	223	LATIN SMALL LETTER SHARP S (German)
à	a!	0xe0	224	LATIN SMALL LETTER A WITH GRAVE
á	a'	0xe1	225	LATIN SMALL LETTER A WITH ACUTE
â	a>	0xe2	226	LATIN SMALL LETTER A WITH CIRCUMFLEX
ã	a?	0xe3	227	LATIN SMALL LETTER A WITH TILDE
ä	a:	0xe4	228	LATIN SMALL LETTER A WITH DIAERESIS
å	aa	0xe5	229	LATIN SMALL LETTER A WITH RING ABOVE
æ	ae	0xe6	230	LATIN SMALL LETTER AE
ç	c,	0xe7	231	LATIN SMALL LETTER C WITH CEDILLA
è	e!	0xe8	232	LATIN SMALL LETTER E WITH GRAVE
é	e'	0xe9	233	LATIN SMALL LETTER E WITH ACUTE
ê	e>	0xea	234	LATIN SMALL LETTER E WITH CIRCUMFLEX
ë	e:	0xeb	235	LATIN SMALL LETTER E WITH DIAERESIS
ì	i!	0xec	236	LATIN SMALL LETTER I WITH GRAVE
í	i'	0xed	237	LATIN SMALL LETTER I WITH ACUTE
î	i>	0xee	238	LATIN SMALL LETTER I WITH CIRCUMFLEX
ï	i:	0xef	239	LATIN SMALL LETTER I WITH DIAERESIS
ð	d-	0xf0	240	LATIN SMALL LETTER ETH (Icelandic)
ñ	n?	0xf1	241	LATIN SMALL LETTER N WITH TILDE
ò	o!	0xf2	242	LATIN SMALL LETTER O WITH GRAVE
ó	o'	0xf3	243	LATIN SMALL LETTER O WITH ACUTE
ô	o>	0xf4	244	LATIN SMALL LETTER O WITH CIRCUMFLEX
õ	o?	0xf5	245	LATIN SMALL LETTER O WITH TILDE
ö	o:	0xf6	246	LATIN SMALL LETTER O WITH DIAERESIS
÷	-:	0xf7	247	DIVISION SIGN
ø	o/	0xf8	248	LATIN SMALL LETTER O WITH STROKE
ù	u!	0xf9	249	LATIN SMALL LETTER U WITH GRAVE

ú	u'	0xfa	250	LATIN SMALL LETTER U WITH ACUTE
û	u>	0xfb	251	LATIN SMALL LETTER U WITH CIRCUMFLEX
ü	u:	0xfc	252	LATIN SMALL LETTER U WITH DIAERESIS
ý	y'	0xfd	253	LATIN SMALL LETTER Y WITH ACUTE
þ	th	0xfe	254	LATIN SMALL LETTER THORN (Icelandic)
ÿ	y:	0xff	255	LATIN SMALL LETTER Y WITH DIAERESIS

If your Vim is compiled with |multibyte| support and you are using a multibyte
'encoding', Vim provides this enhanced set of additional digraphs:

digraph-table-mbyte

char	digraph	hex	dec	official name ~
Ā	A-	0100	0256	LATIN CAPITAL LETTER A WITH MACRON
ā	a-	0101	0257	LATIN SMALL LETTER A WITH MACRON
Ă	A(	0102	0258	LATIN CAPITAL LETTER A WITH BREVE
ă	a(	0103	0259	LATIN SMALL LETTER A WITH BREVE
Ą	A;	0104	0260	LATIN CAPITAL LETTER A WITH OGONEK
ą	a;	0105	0261	LATIN SMALL LETTER A WITH OGONEK
Ć	C'	0106	0262	LATIN CAPITAL LETTER C WITH ACUTE
ć	c'	0107	0263	LATIN SMALL LETTER C WITH ACUTE
Ĉ	C>	0108	0264	LATIN CAPITAL LETTER C WITH CIRCUMFLEX
ĉ	c>	0109	0265	LATIN SMALL LETTER C WITH CIRCUMFLEX
Ċ	C.	010A	0266	LATIN CAPITAL LETTER C WITH DOT ABOVE
ċ	c.	010B	0267	LATIN SMALL LETTER C WITH DOT ABOVE
Č	C<	010C	0268	LATIN CAPITAL LETTER C WITH CARON
č	c<	010D	0269	LATIN SMALL LETTER C WITH CARON
Ď	D<	010E	0270	LATIN CAPITAL LETTER D WITH CARON
ď	d<	010F	0271	LATIN SMALL LETTER D WITH CARON
Đ	D/	0110	0272	LATIN CAPITAL LETTER D WITH STROKE
đ	d/	0111	0273	LATIN SMALL LETTER D WITH STROKE
Ē	E-	0112	0274	LATIN CAPITAL LETTER E WITH MACRON
ē	e-	0113	0275	LATIN SMALL LETTER E WITH MACRON
Ĕ	E(	0114	0276	LATIN CAPITAL LETTER E WITH BREVE
ĕ	e(	0115	0277	LATIN SMALL LETTER E WITH BREVE
Ė	E.	0116	0278	LATIN CAPITAL LETTER E WITH DOT ABOVE
ė	e.	0117	0279	LATIN SMALL LETTER E WITH DOT ABOVE
Ę	E;	0118	0280	LATIN CAPITAL LETTER E WITH OGONEK
ę	e;	0119	0281	LATIN SMALL LETTER E WITH OGONEK
Ě	E<	011A	0282	LATIN CAPITAL LETTER E WITH CARON
ě	e<	011B	0283	LATIN SMALL LETTER E WITH CARON
Ĝ	G>	011C	0284	LATIN CAPITAL LETTER G WITH CIRCUMFLEX
ĝ	g>	011D	0285	LATIN SMALL LETTER G WITH CIRCUMFLEX
Ğ	G(	011E	0286	LATIN CAPITAL LETTER G WITH BREVE
ğ	g(	011F	0287	LATIN SMALL LETTER G WITH BREVE
Ġ	G.	0120	0288	LATIN CAPITAL LETTER G WITH DOT ABOVE
ġ	g.	0121	0289	LATIN SMALL LETTER G WITH DOT ABOVE
Ģ	G,	0122	0290	LATIN CAPITAL LETTER G WITH CEDILLA
ģ	g,	0123	0291	LATIN SMALL LETTER G WITH CEDILLA
Ĥ	H>	0124	0292	LATIN CAPITAL LETTER H WITH CIRCUMFLEX
ĥ	h>	0125	0293	LATIN SMALL LETTER H WITH CIRCUMFLEX
Ħ	H/	0126	0294	LATIN CAPITAL LETTER H WITH STROKE
ħ	h/	0127	0295	LATIN SMALL LETTER H WITH STROKE
Ĩ	I?	0128	0296	LATIN CAPITAL LETTER I WITH TILDE
ĩ	i?	0129	0297	LATIN SMALL LETTER I WITH TILDE
Ī	I-	012A	0298	LATIN CAPITAL LETTER I WITH MACRON
ī	i-	012B	0299	LATIN SMALL LETTER I WITH MACRON
Ĭ	I(	012C	0300	LATIN CAPITAL LETTER I WITH BREVE
ĭ	i(	012D	0301	LATIN SMALL LETTER I WITH BREVE
Į	I;	012E	0302	LATIN CAPITAL LETTER I WITH OGONEK
į	i;	012F	0303	LATIN SMALL LETTER I WITH OGONEK
İ	I.	0130	0304	LATIN CAPITAL LETTER I WITH DOT ABOVE
ı	i.	0131	0305	LATIN SMALL LETTER DOTLESS I

```
Ĳ       IJ      0132    0306    LATIN CAPITAL LIGATURE IJ
ĳ       ij      0133    0307    LATIN SMALL LIGATURE IJ
Ĵ       J>      0134    0308    LATIN CAPITAL LETTER J WITH CIRCUMFLEX
ĵ       j>      0135    0309    LATIN SMALL LETTER J WITH CIRCUMFLEX
Ķ       K,      0136    0310    LATIN CAPITAL LETTER K WITH CEDILLA
ķ       k,      0137    0311    LATIN SMALL LETTER K WITH CEDILLA
ĸ       kk      0138    0312    LATIN SMALL LETTER KRA
Ĺ       L'      0139    0313    LATIN CAPITAL LETTER L WITH ACUTE
ĺ       l'      013A    0314    LATIN SMALL LETTER L WITH ACUTE
Ļ       L,      013B    0315    LATIN CAPITAL LETTER L WITH CEDILLA
ļ       l,      013C    0316    LATIN SMALL LETTER L WITH CEDILLA
Ľ       L<      013D    0317    LATIN CAPITAL LETTER L WITH CARON
ľ       l<      013E    0318    LATIN SMALL LETTER L WITH CARON
Ŀ       L.      013F    0319    LATIN CAPITAL LETTER L WITH MIDDLE DOT
ŀ       l.      0140    0320    LATIN SMALL LETTER L WITH MIDDLE DOT
Ł       L/      0141    0321    LATIN CAPITAL LETTER L WITH STROKE
ł       l/      0142    0322    LATIN SMALL LETTER L WITH STROKE
Ń       N'      0143    0323    LATIN CAPITAL LETTER N WITH ACUTE `
ń       n'      0144    0324    LATIN SMALL LETTER N WITH ACUTE `
Ņ       N,      0145    0325    LATIN CAPITAL LETTER N WITH CEDILLA `
ņ       n,      0146    0326    LATIN SMALL LETTER N WITH CEDILLA `
Ň       N<      0147    0327    LATIN CAPITAL LETTER N WITH CARON `
ň       n<      0148    0328    LATIN SMALL LETTER N WITH CARON `
ŉ       'n      0149    0329    LATIN SMALL LETTER N PRECEDED BY APOSTROPHE `
Ŋ       NG      014A    0330    LATIN CAPITAL LETTER ENG
ŋ       ng      014B    0331    LATIN SMALL LETTER ENG
Ō       O-      014C    0332    LATIN CAPITAL LETTER O WITH MACRON
ō       o-      014D    0333    LATIN SMALL LETTER O WITH MACRON
Ŏ       O(      014E    0334    LATIN CAPITAL LETTER O WITH BREVE
ŏ       o(      014F    0335    LATIN SMALL LETTER O WITH BREVE
Ő       O"      0150    0336    LATIN CAPITAL LETTER O WITH DOUBLE ACUTE
ő       o"      0151    0337    LATIN SMALL LETTER O WITH DOUBLE ACUTE
Œ       OE      0152    0338    LATIN CAPITAL LIGATURE OE
œ       oe      0153    0339    LATIN SMALL LIGATURE OE
Ŕ       R'      0154    0340    LATIN CAPITAL LETTER R WITH ACUTE
ŕ       r'      0155    0341    LATIN SMALL LETTER R WITH ACUTE
Ŗ       R,      0156    0342    LATIN CAPITAL LETTER R WITH CEDILLA
ŗ       r,      0157    0343    LATIN SMALL LETTER R WITH CEDILLA
Ř       R<      0158    0344    LATIN CAPITAL LETTER R WITH CARON
ř       r<      0159    0345    LATIN SMALL LETTER R WITH CARON
Ś       S'      015A    0346    LATIN CAPITAL LETTER S WITH ACUTE
ś       s'      015B    0347    LATIN SMALL LETTER S WITH ACUTE
Ŝ       S>      015C    0348    LATIN CAPITAL LETTER S WITH CIRCUMFLEX
ŝ       s>      015D    0349    LATIN SMALL LETTER S WITH CIRCUMFLEX
Ş       S,      015E    0350    LATIN CAPITAL LETTER S WITH CEDILLA
ş       s,      015F    0351    LATIN SMALL LETTER S WITH CEDILLA
Š       S<      0160    0352    LATIN CAPITAL LETTER S WITH CARON
š       s<      0161    0353    LATIN SMALL LETTER S WITH CARON
Ţ       T,      0162    0354    LATIN CAPITAL LETTER T WITH CEDILLA
ţ       t,      0163    0355    LATIN SMALL LETTER T WITH CEDILLA
Ť       T<      0164    0356    LATIN CAPITAL LETTER T WITH CARON
ť       t<      0165    0357    LATIN SMALL LETTER T WITH CARON
Ŧ       T/      0166    0358    LATIN CAPITAL LETTER T WITH STROKE
ŧ       t/      0167    0359    LATIN SMALL LETTER T WITH STROKE
Ũ       U?      0168    0360    LATIN CAPITAL LETTER U WITH TILDE
ũ       u?      0169    0361    LATIN SMALL LETTER U WITH TILDE
Ū       U-      016A    0362    LATIN CAPITAL LETTER U WITH MACRON
ū       u-      016B    0363    LATIN SMALL LETTER U WITH MACRON
Ŭ       U(      016C    0364    LATIN CAPITAL LETTER U WITH BREVE
ŭ       u(      016D    0365    LATIN SMALL LETTER U WITH BREVE
Ů       U0      016E    0366    LATIN CAPITAL LETTER U WITH RING ABOVE
ů       u0      016F    0367    LATIN SMALL LETTER U WITH RING ABOVE
```

Ű	U"	0170	0368	LATIN CAPITAL LETTER U WITH DOUBLE ACUTE
ű	u"	0171	0369	LATIN SMALL LETTER U WITH DOUBLE ACUTE
Ų	U;	0172	0370	LATIN CAPITAL LETTER U WITH OGONEK
ų	u;	0173	0371	LATIN SMALL LETTER U WITH OGONEK
Ŵ	W>	0174	0372	LATIN CAPITAL LETTER W WITH CIRCUMFLEX
ŵ	w>	0175	0373	LATIN SMALL LETTER W WITH CIRCUMFLEX
Ŷ	Y>	0176	0374	LATIN CAPITAL LETTER Y WITH CIRCUMFLEX
ŷ	y>	0177	0375	LATIN SMALL LETTER Y WITH CIRCUMFLEX
Ÿ	Y:	0178	0376	LATIN CAPITAL LETTER Y WITH DIAERESIS
Ź	Z'	0179	0377	LATIN CAPITAL LETTER Z WITH ACUTE
ź	z'	017A	0378	LATIN SMALL LETTER Z WITH ACUTE
Ż	Z.	017B	0379	LATIN CAPITAL LETTER Z WITH DOT ABOVE
ż	z.	017C	0380	LATIN SMALL LETTER Z WITH DOT ABOVE
Ž	Z<	017D	0381	LATIN CAPITAL LETTER Z WITH CARON
ž	z<	017E	0382	LATIN SMALL LETTER Z WITH CARON
Ơ	O9	01A0	0416	LATIN CAPITAL LETTER O WITH HORN
ơ	o9	01A1	0417	LATIN SMALL LETTER O WITH HORN
Ƣ	OI	01A2	0418	LATIN CAPITAL LETTER OI
ƣ	oi	01A3	0419	LATIN SMALL LETTER OI
Ʀ	yr	01A6	0422	LATIN LETTER YR
Ư	U9	01AF	0431	LATIN CAPITAL LETTER U WITH HORN
ư	u9	01B0	0432	LATIN SMALL LETTER U WITH HORN
Ƶ	Z/	01B5	0437	LATIN CAPITAL LETTER Z WITH STROKE
ƶ	z/	01B6	0438	LATIN SMALL LETTER Z WITH STROKE
Ʒ	ED	01B7	0439	LATIN CAPITAL LETTER EZH
Ǎ	A<	01CD	0461	LATIN CAPITAL LETTER A WITH CARON
ǎ	a<	01CE	0462	LATIN SMALL LETTER A WITH CARON
Ǐ	I<	01CF	0463	LATIN CAPITAL LETTER I WITH CARON
ǐ	i<	01D0	0464	LATIN SMALL LETTER I WITH CARON
Ǒ	O<	01D1	0465	LATIN CAPITAL LETTER O WITH CARON
ǒ	o<	01D2	0466	LATIN SMALL LETTER O WITH CARON
Ǔ	U<	01D3	0467	LATIN CAPITAL LETTER U WITH CARON
ǔ	u<	01D4	0468	LATIN SMALL LETTER U WITH CARON
Ǟ	A1	01DE	0478	LATIN CAPITAL LETTER A WITH DIAERESIS AND MACRON
ǟ	a1	01DF	0479	LATIN SMALL LETTER A WITH DIAERESIS AND MACRON
Ǡ	A7	01E0	0480	LATIN CAPITAL LETTER A WITH DOT ABOVE AND MACRON
ǡ	a7	01E1	0481	LATIN SMALL LETTER A WITH DOT ABOVE AND MACRON
Ǣ	A3	01E2	0482	LATIN CAPITAL LETTER AE WITH MACRON
ǣ	a3	01E3	0483	LATIN SMALL LETTER AE WITH MACRON
Ǥ	G/	01E4	0484	LATIN CAPITAL LETTER G WITH STROKE
ǥ	g/	01E5	0485	LATIN SMALL LETTER G WITH STROKE
Ǧ	G<	01E6	0486	LATIN CAPITAL LETTER G WITH CARON
ǧ	g<	01E7	0487	LATIN SMALL LETTER G WITH CARON
Ǩ	K<	01E8	0488	LATIN CAPITAL LETTER K WITH CARON
ǩ	k<	01E9	0489	LATIN SMALL LETTER K WITH CARON
Ǫ	O;	01EA	0490	LATIN CAPITAL LETTER O WITH OGONEK
ǫ	o;	01EB	0491	LATIN SMALL LETTER O WITH OGONEK
Ǭ	O1	01EC	0492	LATIN CAPITAL LETTER O WITH OGONEK AND MACRON
ǭ	o1	01ED	0493	LATIN SMALL LETTER O WITH OGONEK AND MACRON
Ǯ	EZ	01EE	0494	LATIN CAPITAL LETTER EZH WITH CARON
ǯ	ez	01EF	0495	LATIN SMALL LETTER EZH WITH CARON
ǰ	j<	01F0	0496	LATIN SMALL LETTER J WITH CARON
Ǵ	G'	01F4	0500	LATIN CAPITAL LETTER G WITH ACUTE
ǵ	g'	01F5	0501	LATIN SMALL LETTER G WITH ACUTE
ʿ	;S	02BF	0703	MODIFIER LETTER LEFT HALF RING
ˇ	'<	02C7	0711	CARON
˘	'(	02D8	0728	BREVE
˙	'.	02D9	0729	DOT ABOVE
˚	'0	02DA	0730	RING ABOVE
˛	';	02DB	0731	OGONEK
˝	'"	02DD	0733	DOUBLE ACUTE ACCENT
Ά	A%	0386	0902	GREEK CAPITAL LETTER ALPHA WITH TONOS

Έ	E%	0388	0904	GREEK CAPITAL LETTER EPSILON WITH TONOS
Ή	Y%	0389	0905	GREEK CAPITAL LETTER ETA WITH TONOS
Ί	I%	038A	0906	GREEK CAPITAL LETTER IOTA WITH TONOS
Ό	O%	038C	0908	GREEK CAPITAL LETTER OMICRON WITH TONOS
Ύ	U%	038E	0910	GREEK CAPITAL LETTER UPSILON WITH TONOS
Ώ	W%	038F	0911	GREEK CAPITAL LETTER OMEGA WITH TONOS
ΐ	i3	0390	0912	GREEK SMALL LETTER IOTA WITH DIALYTIKA AND TONOS
Α	A*	0391	0913	GREEK CAPITAL LETTER ALPHA
Β	B*	0392	0914	GREEK CAPITAL LETTER BETA
Γ	G*	0393	0915	GREEK CAPITAL LETTER GAMMA
Δ	D*	0394	0916	GREEK CAPITAL LETTER DELTA
Ε	E*	0395	0917	GREEK CAPITAL LETTER EPSILON
Ζ	Z*	0396	0918	GREEK CAPITAL LETTER ZETA
Η	Y*	0397	0919	GREEK CAPITAL LETTER ETA
Θ	H*	0398	0920	GREEK CAPITAL LETTER THETA
Ι	I*	0399	0921	GREEK CAPITAL LETTER IOTA
Κ	K*	039A	0922	GREEK CAPITAL LETTER KAPPA
Λ	L*	039B	0923	GREEK CAPITAL LETTER LAMDA
Μ	M*	039C	0924	GREEK CAPITAL LETTER MU
Ν	N*	039D	0925	GREEK CAPITAL LETTER NU
Ξ	C*	039E	0926	GREEK CAPITAL LETTER XI
Ο	O*	039F	0927	GREEK CAPITAL LETTER OMICRON
Π	P*	03A0	0928	GREEK CAPITAL LETTER PI
Ρ	R*	03A1	0929	GREEK CAPITAL LETTER RHO
Σ	S*	03A3	0931	GREEK CAPITAL LETTER SIGMA
Τ	T*	03A4	0932	GREEK CAPITAL LETTER TAU
Υ	U*	03A5	0933	GREEK CAPITAL LETTER UPSILON
Φ	F*	03A6	0934	GREEK CAPITAL LETTER PHI
Χ	X*	03A7	0935	GREEK CAPITAL LETTER CHI
Ψ	Q*	03A8	0936	GREEK CAPITAL LETTER PSI
Ω	W*	03A9	0937	GREEK CAPITAL LETTER OMEGA
Ϊ	J*	03AA	0938	GREEK CAPITAL LETTER IOTA WITH DIALYTIKA
Ϋ	V*	03AB	0939	GREEK CAPITAL LETTER UPSILON WITH DIALYTIKA
ά	a%	03AC	0940	GREEK SMALL LETTER ALPHA WITH TONOS
έ	e%	03AD	0941	GREEK SMALL LETTER EPSILON WITH TONOS
ή	y%	03AE	0942	GREEK SMALL LETTER ETA WITH TONOS
ί	i%	03AF	0943	GREEK SMALL LETTER IOTA WITH TONOS
ΰ	u3	03B0	0944	GREEK SMALL LETTER UPSILON WITH DIALYTIKA AND TONOS
α	a*	03B1	0945	GREEK SMALL LETTER ALPHA
β	b*	03B2	0946	GREEK SMALL LETTER BETA
γ	g*	03B3	0947	GREEK SMALL LETTER GAMMA
δ	d*	03B4	0948	GREEK SMALL LETTER DELTA
ε	e*	03B5	0949	GREEK SMALL LETTER EPSILON
ζ	z*	03B6	0950	GREEK SMALL LETTER ZETA
η	y*	03B7	0951	GREEK SMALL LETTER ETA
θ	h*	03B8	0952	GREEK SMALL LETTER THETA
ι	i*	03B9	0953	GREEK SMALL LETTER IOTA
κ	k*	03BA	0954	GREEK SMALL LETTER KAPPA
λ	l*	03BB	0955	GREEK SMALL LETTER LAMDA
μ	m*	03BC	0956	GREEK SMALL LETTER MU
ν	n*	03BD	0957	GREEK SMALL LETTER NU
ξ	c*	03BE	0958	GREEK SMALL LETTER XI
ο	o*	03BF	0959	GREEK SMALL LETTER OMICRON
π	p*	03C0	0960	GREEK SMALL LETTER PI
ρ	r*	03C1	0961	GREEK SMALL LETTER RHO
ς	*s	03C2	0962	GREEK SMALL LETTER FINAL SIGMA
σ	s*	03C3	0963	GREEK SMALL LETTER SIGMA
τ	t*	03C4	0964	GREEK SMALL LETTER TAU
υ	u*	03C5	0965	GREEK SMALL LETTER UPSILON
φ	f*	03C6	0966	GREEK SMALL LETTER PHI
χ	x*	03C7	0967	GREEK SMALL LETTER CHI
ψ	q*	03C8	0968	GREEK SMALL LETTER PSI

ω	w*	03C9	0969	GREEK SMALL LETTER OMEGA
ϊ	j*	03CA	0970	GREEK SMALL LETTER IOTA WITH DIALYTIKA
ϋ	v*	03CB	0971	GREEK SMALL LETTER UPSILON WITH DIALYTIKA
ό	o%	03CC	0972	GREEK SMALL LETTER OMICRON WITH TONOS
ύ	u%	03CD	0973	GREEK SMALL LETTER UPSILON WITH TONOS
ώ	w%	03CE	0974	GREEK SMALL LETTER OMEGA WITH TONOS
Ϙ	'G	03D8	0984	GREEK LETTER ARCHAIC KOPPA
ϙ	,G	03D9	0985	GREEK SMALL LETTER ARCHAIC KOPPA
Ϛ	T3	03DA	0986	GREEK LETTER STIGMA
ϛ	t3	03DB	0987	GREEK SMALL LETTER STIGMA
Ϝ	M3	03DC	0988	GREEK LETTER DIGAMMA
ϝ	m3	03DD	0989	GREEK SMALL LETTER DIGAMMA
Ϟ	K3	03DE	0990	GREEK LETTER KOPPA
ϟ	k3	03DF	0991	GREEK SMALL LETTER KOPPA
Ϡ	P3	03E0	0992	GREEK LETTER SAMPI
ϡ	p3	03E1	0993	GREEK SMALL LETTER SAMPI
ϴ	'%	03F4	1012	GREEK CAPITAL THETA SYMBOL
ϵ	j3	03F5	1013	GREEK LUNATE EPSILON SYMBOL
Ё	IO	0401	1025	CYRILLIC CAPITAL LETTER IO
Ђ	D%	0402	1026	CYRILLIC CAPITAL LETTER DJE
Ѓ	G%	0403	1027	CYRILLIC CAPITAL LETTER GJE
Є	IE	0404	1028	CYRILLIC CAPITAL LETTER UKRAINIAN IE
Ѕ	DS	0405	1029	CYRILLIC CAPITAL LETTER DZE
І	II	0406	1030	CYRILLIC CAPITAL LETTER BYELORUSSIAN-UKRAINIAN I
Ї	YI	0407	1031	CYRILLIC CAPITAL LETTER YI
Ј	J%	0408	1032	CYRILLIC CAPITAL LETTER JE
Љ	LJ	0409	1033	CYRILLIC CAPITAL LETTER LJE
Њ	NJ	040A	1034	CYRILLIC CAPITAL LETTER NJE
Ћ	Ts	040B	1035	CYRILLIC CAPITAL LETTER TSHE
Ќ	KJ	040C	1036	CYRILLIC CAPITAL LETTER KJE
Ў	V%	040E	1038	CYRILLIC CAPITAL LETTER SHORT U
Џ	DZ	040F	1039	CYRILLIC CAPITAL LETTER DZHE
А	A=	0410	1040	CYRILLIC CAPITAL LETTER A
Б	B=	0411	1041	CYRILLIC CAPITAL LETTER BE
В	V=	0412	1042	CYRILLIC CAPITAL LETTER VE
Г	G=	0413	1043	CYRILLIC CAPITAL LETTER GHE
Д	D=	0414	1044	CYRILLIC CAPITAL LETTER DE
Е	E=	0415	1045	CYRILLIC CAPITAL LETTER IE
Ж	Z%	0416	1046	CYRILLIC CAPITAL LETTER ZHE
З	Z=	0417	1047	CYRILLIC CAPITAL LETTER ZE
И	I=	0418	1048	CYRILLIC CAPITAL LETTER I
Й	J=	0419	1049	CYRILLIC CAPITAL LETTER SHORT I
К	K=	041A	1050	CYRILLIC CAPITAL LETTER KA
Л	L=	041B	1051	CYRILLIC CAPITAL LETTER EL
М	M=	041C	1052	CYRILLIC CAPITAL LETTER EM
Н	N=	041D	1053	CYRILLIC CAPITAL LETTER EN
О	O=	041E	1054	CYRILLIC CAPITAL LETTER O
П	P=	041F	1055	CYRILLIC CAPITAL LETTER PE
Р	R=	0420	1056	CYRILLIC CAPITAL LETTER ER
С	S=	0421	1057	CYRILLIC CAPITAL LETTER ES
Т	T=	0422	1058	CYRILLIC CAPITAL LETTER TE
У	U=	0423	1059	CYRILLIC CAPITAL LETTER U
Ф	F=	0424	1060	CYRILLIC CAPITAL LETTER EF
Х	H=	0425	1061	CYRILLIC CAPITAL LETTER HA
Ц	C=	0426	1062	CYRILLIC CAPITAL LETTER TSE
Ч	C%	0427	1063	CYRILLIC CAPITAL LETTER CHE
Ш	S%	0428	1064	CYRILLIC CAPITAL LETTER SHA
Щ	Sc	0429	1065	CYRILLIC CAPITAL LETTER SHCHA
Ъ	="	042A	1066	CYRILLIC CAPITAL LETTER HARD SIGN
Ы	Y=	042B	1067	CYRILLIC CAPITAL LETTER YERU
Ь	%"	042C	1068	CYRILLIC CAPITAL LETTER SOFT SIGN
Э	JE	042D	1069	CYRILLIC CAPITAL LETTER E

```
Ю       JU      042E    1070    CYRILLIC CAPITAL LETTER YU
Я       JA      042F    1071    CYRILLIC CAPITAL LETTER YA
а       a=      0430    1072    CYRILLIC SMALL LETTER A
б       b=      0431    1073    CYRILLIC SMALL LETTER BE
в       v=      0432    1074    CYRILLIC SMALL LETTER VE
г       g=      0433    1075    CYRILLIC SMALL LETTER GHE
д       d=      0434    1076    CYRILLIC SMALL LETTER DE
е       e=      0435    1077    CYRILLIC SMALL LETTER IE
ж       z%      0436    1078    CYRILLIC SMALL LETTER ZHE
з       z=      0437    1079    CYRILLIC SMALL LETTER ZE
и       i=      0438    1080    CYRILLIC SMALL LETTER I
й       j=      0439    1081    CYRILLIC SMALL LETTER SHORT I
к       k=      043A    1082    CYRILLIC SMALL LETTER KA
л       l=      043B    1083    CYRILLIC SMALL LETTER EL
м       m=      043C    1084    CYRILLIC SMALL LETTER EM
н       n=      043D    1085    CYRILLIC SMALL LETTER EN
о       o=      043E    1086    CYRILLIC SMALL LETTER O
п       p=      043F    1087    CYRILLIC SMALL LETTER PE
р       r=      0440    1088    CYRILLIC SMALL LETTER ER
с       s=      0441    1089    CYRILLIC SMALL LETTER ES
т       t=      0442    1090    CYRILLIC SMALL LETTER TE
у       u=      0443    1091    CYRILLIC SMALL LETTER U
ф       f=      0444    1092    CYRILLIC SMALL LETTER EF
х       h=      0445    1093    CYRILLIC SMALL LETTER HA
ц       c=      0446    1094    CYRILLIC SMALL LETTER TSE
ч       c%      0447    1095    CYRILLIC SMALL LETTER CHE
ш       s%      0448    1096    CYRILLIC SMALL LETTER SHA
щ       sc      0449    1097    CYRILLIC SMALL LETTER SHCHA
ъ       ='      044A    1098    CYRILLIC SMALL LETTER HARD SIGN
ы       y=      044B    1099    CYRILLIC SMALL LETTER YERU
ь       %'      044C    1100    CYRILLIC SMALL LETTER SOFT SIGN
э       je      044D    1101    CYRILLIC SMALL LETTER E
ю       ju      044E    1102    CYRILLIC SMALL LETTER YU
я       ja      044F    1103    CYRILLIC SMALL LETTER YA
ё       io      0451    1105    CYRILLIC SMALL LETTER IO
ђ       d%      0452    1106    CYRILLIC SMALL LETTER DJE
ѓ       g%      0453    1107    CYRILLIC SMALL LETTER GJE
є       ie      0454    1108    CYRILLIC SMALL LETTER UKRAINIAN IE
ѕ       ds      0455    1109    CYRILLIC SMALL LETTER DZE
і       ii      0456    1110    CYRILLIC SMALL LETTER BYELORUSSIAN-UKRAINIAN I
ї       yi      0457    1111    CYRILLIC SMALL LETTER YI
ј       j%      0458    1112    CYRILLIC SMALL LETTER JE
љ       lj      0459    1113    CYRILLIC SMALL LETTER LJE
њ       nj      045A    1114    CYRILLIC SMALL LETTER NJE
ћ       ts      045B    1115    CYRILLIC SMALL LETTER TSHE
ќ       kj      045C    1116    CYRILLIC SMALL LETTER KJE
ў       v%      045E    1118    CYRILLIC SMALL LETTER SHORT U
џ       dz      045F    1119    CYRILLIC SMALL LETTER DZHE
Ѣ       Y3      0462    1122    CYRILLIC CAPITAL LETTER YAT
ѣ       y3      0463    1123    CYRILLIC SMALL LETTER YAT
Ѫ       O3      046A    1130    CYRILLIC CAPITAL LETTER BIG YUS
ѫ       o3      046B    1131    CYRILLIC SMALL LETTER BIG YUS
Ѳ       F3      0472    1138    CYRILLIC CAPITAL LETTER FITA
ѳ       f3      0473    1139    CYRILLIC SMALL LETTER FITA
Ѵ       V3      0474    1140    CYRILLIC CAPITAL LETTER IZHITSA
ѵ       v3      0475    1141    CYRILLIC SMALL LETTER IZHITSA
Ҁ       C3      0480    1152    CYRILLIC CAPITAL LETTER KOPPA
ҁ       c3      0481    1153    CYRILLIC SMALL LETTER KOPPA
Ґ       G3      0490    1168    CYRILLIC CAPITAL LETTER GHE WITH UPTURN
ґ       g3      0491    1169    CYRILLIC SMALL LETTER GHE WITH UPTURN
                                     HEBREW LETTER ALEF    1488    05D0      A+       א
```

```
                                      HEBREW LETTER BET    1489    05D1     B+       ב
                                    HEBREW LETTER GIMEL    1490    05D2     G+       ג
                                    HEBREW LETTER DALET    1491    05D3     D+       ד
                                       HEBREW LETTER HE    1492    05D4     H+       ה
                                      HEBREW LETTER VAV    1493    05D5     W+       ו
                                    HEBREW LETTER ZAYIN    1494    05D6     Z+       ז
                                      HEBREW LETTER HET    1495    05D7     X+       ח
                                      HEBREW LETTER TET    1496    05D8     Tj       ט
                                      HEBREW LETTER YOD    1497    05D9     J+       י
                                HEBREW LETTER FINAL KAF    1498    05DA     K%       ך
                                      HEBREW LETTER KAF    1499    05DB     K+       כ
                                    HEBREW LETTER LAMED    1500    05DC     L+       ל
                                HEBREW LETTER FINAL MEM    1501    05DD     M%       ם
                                      HEBREW LETTER MEM    1502    05DE     M+       מ
                              ` HEBREW LETTER FINAL NUN    1503    05DF     N%       ן
                                    ` HEBREW LETTER NUN    1504    05E0     N+       נ
                                   HEBREW LETTER SAMEKH    1505    05E1     S+       ס
                                     HEBREW LETTER AYIN    1506    05E2     E+       ע
                                 HEBREW LETTER FINAL PE    1507    05E3     P%       ף
                                       HEBREW LETTER PE    1508    05E4     P+       פ
                              HEBREW LETTER FINAL TSADI    1509    05E5     Zj       ץ
                                    HEBREW LETTER TSADI    1510    05E6     ZJ       צ
                                      HEBREW LETTER QOF    1511    05E7     Q+       ק
                                     HEBREW LETTER RESH    1512    05E8     R+       ר
                                     HEBREW LETTER SHIN    1513    05E9     Sh       ש
                                      HEBREW LETTER TAV    1514    05EA     T+       ת
،       ,+      060C    1548    ARABIC COMMA
                                       ARABIC SEMICOLON    1563    061B     +;       ؛
                                   ARABIC QUESTION MARK    1567    061F     +?       ؟
                                    ARABIC LETTER HAMZA    1569    0621     H'       ء
                    ARABIC LETTER ALEF WITH MADDA ABOVE    1570    0622     aM       آ
                    ARABIC LETTER ALEF WITH HAMZA ABOVE    1571    0623     aH       أ
                     ARABIC LETTER WAW WITH HAMZA ABOVE    1572    0624     wH       ؤ
                    ARABIC LETTER ALEF WITH HAMZA BELOW    1573    0625     ah       إ
                     ARABIC LETTER YEH WITH HAMZA ABOVE    1574    0626     yH       ئ
                                     ARABIC LETTER ALEF    1575    0627     a+       ا
                                      ARABIC LETTER BEH    1576    0628     b+       ب
                              ARABIC LETTER TEH MARBUTA    1577    0629     tm       ة
                                      ARABIC LETTER TEH    1578    062A     t+       ت
                                     ARABIC LETTER THEH    1579    062B     tk       ث
                                     ARABIC LETTER JEEM    1580    062C     g+       ج
                                      ARABIC LETTER HAH    1581    062D     hk       ح
                                     ARABIC LETTER KHAH    1582    062E     x+       خ
                                      ARABIC LETTER DAL    1583    062F     d+       د
                                     ARABIC LETTER THAL    1584    0630     dk       ذ
                                      ARABIC LETTER REH    1585    0631     r+       ر
                                     ARABIC LETTER ZAIN    1586    0632     z+       ز
                                     ARABIC LETTER SEEN    1587    0633     s+       س
                                    ARABIC LETTER SHEEN    1588    0634     sn       ش
                                      ARABIC LETTER SAD    1589    0635     c+       ص
                                      ARABIC LETTER DAD    1590    0636     dd       ض
                                      ARABIC LETTER TAH    1591    0637     tj       ط
                                      ARABIC LETTER ZAH    1592    0638     zH       ظ
                                      ARABIC LETTER AIN    1593    0639     e+       ع
                                    ARABIC LETTER GHAIN    1594    063A     i+       غ
                                         ARABIC TATWEEL    1600    0640     ++       ـ
                                      ARABIC LETTER FEH    1601    0641     f+       ف
                                      ARABIC LETTER QAF    1602    0642     q+       ق
                                      ARABIC LETTER KAF    1603    0643     k+       ك
                                      ARABIC LETTER LAM    1604    0644     l+       ل
                                     ARABIC LETTER MEEM    1605    0645     m+       م
                                     ARABIC LETTER NOON    1606    0646     n+       ن
```

```
                                      ARABIC LETTER HEH    1607    0647      h+        ه
                                      ARABIC LETTER WAW    1608    0648      w+        و
                             ARABIC LETTER ALEF MAKSURA    1609    0649      j+        ى
                                      ARABIC LETTER YEH    1610    064A      y+        ي
◌ً       :+      064B    1611    ARABIC FATHATAN
◌ٌ       "+      064C    1612    ARABIC DAMMATAN
◌ٍ       =+      064D    1613    ARABIC KASRATAN
◌َ       /+      064E    1614    ARABIC FATHA
◌ُ       '+      064F    1615    ARABIC DAMMA
◌ِ       1+      0650    1616    ARABIC KASRA
◌ّ       3+      0651    1617    ARABIC SHADDA
◌ْ       0+      0652    1618    ARABIC SUKUN
◌ٰ       aS      0670    1648    ARABIC LETTER SUPERSCRIPT ALEF
                                      ARABIC LETTER PEH    1662    067E      p+        پ
                                      ARABIC LETTER VEH    1700    06A4      v+        ڤ
                                      ARABIC LETTER GAF    1711    06AF      gf        گ
۰       0a      06F0    1776    EXTENDED ARABIC-INDIC DIGIT ZERO
۱       1a      06F1    1777    EXTENDED ARABIC-INDIC DIGIT ONE
۲       2a      06F2    1778    EXTENDED ARABIC-INDIC DIGIT TWO
۳       3a      06F3    1779    EXTENDED ARABIC-INDIC DIGIT THREE
۴       4a      06F4    1780    EXTENDED ARABIC-INDIC DIGIT FOUR
۵       5a      06F5    1781    EXTENDED ARABIC-INDIC DIGIT FIVE
۶       6a      06F6    1782    EXTENDED ARABIC-INDIC DIGIT SIX
۷       7a      06F7    1783    EXTENDED ARABIC-INDIC DIGIT SEVEN
۸       8a      06F8    1784    EXTENDED ARABIC-INDIC DIGIT EIGHT
۹       9a      06F9    1785    EXTENDED ARABIC-INDIC DIGIT NINE
Ḃ       B.      1E02    7682    LATIN CAPITAL LETTER B WITH DOT ABOVE
ḃ       b.      1E03    7683    LATIN SMALL LETTER B WITH DOT ABOVE
Ḇ       B_      1E06    7686    LATIN CAPITAL LETTER B WITH LINE BELOW
ḇ       b_      1E07    7687    LATIN SMALL LETTER B WITH LINE BELOW
Ḋ       D.      1E0A    7690    LATIN CAPITAL LETTER D WITH DOT ABOVE
ḋ       d.      1E0B    7691    LATIN SMALL LETTER D WITH DOT ABOVE
Ḏ       D_      1E0E    7694    LATIN CAPITAL LETTER D WITH LINE BELOW
ḏ       d_      1E0F    7695    LATIN SMALL LETTER D WITH LINE BELOW
Ḑ       D,      1E10    7696    LATIN CAPITAL LETTER D WITH CEDILLA
ḑ       d,      1E11    7697    LATIN SMALL LETTER D WITH CEDILLA
Ḟ       F.      1E1E    7710    LATIN CAPITAL LETTER F WITH DOT ABOVE
ḟ       f.      1E1F    7711    LATIN SMALL LETTER F WITH DOT ABOVE
Ḡ       G-      1E20    7712    LATIN CAPITAL LETTER G WITH MACRON
ḡ       g-      1E21    7713    LATIN SMALL LETTER G WITH MACRON
Ḣ       H.      1E22    7714    LATIN CAPITAL LETTER H WITH DOT ABOVE
ḣ       h.      1E23    7715    LATIN SMALL LETTER H WITH DOT ABOVE
Ḧ       H:      1E26    7718    LATIN CAPITAL LETTER H WITH DIAERESIS
ḧ       h:      1E27    7719    LATIN SMALL LETTER H WITH DIAERESIS
Ḩ       H,      1E28    7720    LATIN CAPITAL LETTER H WITH CEDILLA
ḩ       h,      1E29    7721    LATIN SMALL LETTER H WITH CEDILLA
Ḱ       K'      1E30    7728    LATIN CAPITAL LETTER K WITH ACUTE
ḱ       k'      1E31    7729    LATIN SMALL LETTER K WITH ACUTE
Ḵ       K_      1E34    7732    LATIN CAPITAL LETTER K WITH LINE BELOW
ḵ       k_      1E35    7733    LATIN SMALL LETTER K WITH LINE BELOW
Ḻ       L_      1E3A    7738    LATIN CAPITAL LETTER L WITH LINE BELOW
ḻ       l_      1E3B    7739    LATIN SMALL LETTER L WITH LINE BELOW
Ḿ       M'      1E3E    7742    LATIN CAPITAL LETTER M WITH ACUTE
ḿ       m'      1E3F    7743    LATIN SMALL LETTER M WITH ACUTE
Ṁ       M.      1E40    7744    LATIN CAPITAL LETTER M WITH DOT ABOVE
ṁ       m.      1E41    7745    LATIN SMALL LETTER M WITH DOT ABOVE
Ṅ       N.      1E44    7748    LATIN CAPITAL LETTER N WITH DOT ABOVE `
ṅ       n.      1E45    7749    LATIN SMALL LETTER N WITH DOT ABOVE `
Ṉ       N_      1E48    7752    LATIN CAPITAL LETTER N WITH LINE BELOW `
ṉ       n_      1E49    7753    LATIN SMALL LETTER N WITH LINE BELOW `
Ṕ       P'      1E54    7764    LATIN CAPITAL LETTER P WITH ACUTE
ṕ       p'      1E55    7765    LATIN SMALL LETTER P WITH ACUTE
```

Ṗ	P.	1E56	7766	LATIN CAPITAL LETTER P WITH DOT ABOVE
ṗ	p.	1E57	7767	LATIN SMALL LETTER P WITH DOT ABOVE
Ṙ	R.	1E58	7768	LATIN CAPITAL LETTER R WITH DOT ABOVE
ṙ	r.	1E59	7769	LATIN SMALL LETTER R WITH DOT ABOVE
Ṟ	R_	1E5E	7774	LATIN CAPITAL LETTER R WITH LINE BELOW
ṟ	r_	1E5F	7775	LATIN SMALL LETTER R WITH LINE BELOW
Ṡ	S.	1E60	7776	LATIN CAPITAL LETTER S WITH DOT ABOVE
ṡ	s.	1E61	7777	LATIN SMALL LETTER S WITH DOT ABOVE
Ṫ	T.	1E6A	7786	LATIN CAPITAL LETTER T WITH DOT ABOVE
ṫ	t.	1E6B	7787	LATIN SMALL LETTER T WITH DOT ABOVE
Ṯ	T_	1E6E	7790	LATIN CAPITAL LETTER T WITH LINE BELOW
ṯ	t_	1E6F	7791	LATIN SMALL LETTER T WITH LINE BELOW
Ṽ	V?	1E7C	7804	LATIN CAPITAL LETTER V WITH TILDE
ṽ	v?	1E7D	7805	LATIN SMALL LETTER V WITH TILDE
Ẁ	W!	1E80	7808	LATIN CAPITAL LETTER W WITH GRAVE
ẁ	w!	1E81	7809	LATIN SMALL LETTER W WITH GRAVE
Ẃ	W'	1E82	7810	LATIN CAPITAL LETTER W WITH ACUTE
ẃ	w'	1E83	7811	LATIN SMALL LETTER W WITH ACUTE
Ẅ	W:	1E84	7812	LATIN CAPITAL LETTER W WITH DIAERESIS
ẅ	w:	1E85	7813	LATIN SMALL LETTER W WITH DIAERESIS
Ẇ	W.	1E86	7814	LATIN CAPITAL LETTER W WITH DOT ABOVE
ẇ	w.	1E87	7815	LATIN SMALL LETTER W WITH DOT ABOVE
Ẋ	X.	1E8A	7818	LATIN CAPITAL LETTER X WITH DOT ABOVE
ẋ	x.	1E8B	7819	LATIN SMALL LETTER X WITH DOT ABOVE
Ẍ	X:	1E8C	7820	LATIN CAPITAL LETTER X WITH DIAERESIS
ẍ	x:	1E8D	7821	LATIN SMALL LETTER X WITH DIAERESIS
Ẏ	Y.	1E8E	7822	LATIN CAPITAL LETTER Y WITH DOT ABOVE
ẏ	y.	1E8F	7823	LATIN SMALL LETTER Y WITH DOT ABOVE
Ẑ	Z>	1E90	7824	LATIN CAPITAL LETTER Z WITH CIRCUMFLEX
ẑ	z>	1E91	7825	LATIN SMALL LETTER Z WITH CIRCUMFLEX
Ẕ	Z_	1E94	7828	LATIN CAPITAL LETTER Z WITH LINE BELOW
ẕ	z_	1E95	7829	LATIN SMALL LETTER Z WITH LINE BELOW
ẖ	h_	1E96	7830	LATIN SMALL LETTER H WITH LINE BELOW
ẗ	t:	1E97	7831	LATIN SMALL LETTER T WITH DIAERESIS
ẘ	w0	1E98	7832	LATIN SMALL LETTER W WITH RING ABOVE
ẙ	y0	1E99	7833	LATIN SMALL LETTER Y WITH RING ABOVE
Ả	A2	1EA2	7842	LATIN CAPITAL LETTER A WITH HOOK ABOVE
ả	a2	1EA3	7843	LATIN SMALL LETTER A WITH HOOK ABOVE
Ẻ	E2	1EBA	7866	LATIN CAPITAL LETTER E WITH HOOK ABOVE
ẻ	e2	1EBB	7867	LATIN SMALL LETTER E WITH HOOK ABOVE
Ẽ	E?	1EBC	7868	LATIN CAPITAL LETTER E WITH TILDE
ẽ	e?	1EBD	7869	LATIN SMALL LETTER E WITH TILDE
Ỉ	I2	1EC8	7880	LATIN CAPITAL LETTER I WITH HOOK ABOVE
ỉ	i2	1EC9	7881	LATIN SMALL LETTER I WITH HOOK ABOVE
Ỏ	O2	1ECE	7886	LATIN CAPITAL LETTER O WITH HOOK ABOVE
ỏ	o2	1ECF	7887	LATIN SMALL LETTER O WITH HOOK ABOVE
Ủ	U2	1EE6	7910	LATIN CAPITAL LETTER U WITH HOOK ABOVE
ủ	u2	1EE7	7911	LATIN SMALL LETTER U WITH HOOK ABOVE
Ỳ	Y!	1EF2	7922	LATIN CAPITAL LETTER Y WITH GRAVE
ỳ	y!	1EF3	7923	LATIN SMALL LETTER Y WITH GRAVE
Ỷ	Y2	1EF6	7926	LATIN CAPITAL LETTER Y WITH HOOK ABOVE
ỷ	y2	1EF7	7927	LATIN SMALL LETTER Y WITH HOOK ABOVE
Ỹ	Y?	1EF8	7928	LATIN CAPITAL LETTER Y WITH TILDE
ỹ	y?	1EF9	7929	LATIN SMALL LETTER Y WITH TILDE
ἀ	;'	1F00	7936	GREEK SMALL LETTER ALPHA WITH PSILI
ἁ	,'	1F01	7937	GREEK SMALL LETTER ALPHA WITH DASIA
ἂ	;!	1F02	7938	GREEK SMALL LETTER ALPHA WITH PSILI AND VARIA
ἃ	,!	1F03	7939	GREEK SMALL LETTER ALPHA WITH DASIA AND VARIA
ἄ	?;	1F04	7940	GREEK SMALL LETTER ALPHA WITH PSILI AND OXIA
ἅ	?,	1F05	7941	GREEK SMALL LETTER ALPHA WITH DASIA AND OXIA
ἆ	!:	1F06	7942	GREEK SMALL LETTER ALPHA WITH PSILI AND PERISPOMENI

```
ἇ       ?:      1F07    7943    GREEK SMALL LETTER ALPHA WITH DASIA AND PERISPOMENI
        1N      2002    8194    EN SPACE
        1M      2003    8195    EM SPACE
        3M      2004    8196    THREE-PER-EM SPACE
        4M      2005    8197    FOUR-PER-EM SPACE
        6M      2006    8198    SIX-PER-EM SPACE
        1T      2009    8201    THIN SPACE
        1H      200A    8202    HAIR SPACE
‐       -1      2010    8208    HYPHEN
–       -N      2013    8211    EN DASH `
—       -M      2014    8212    EM DASH
―       -3      2015    8213    HORIZONTAL BAR
‖       !2      2016    8214    DOUBLE VERTICAL LINE
‗       =2      2017    8215    DOUBLE LOW LINE
‘       '6      2018    8216    LEFT SINGLE QUOTATION MARK
’       '9      2019    8217    RIGHT SINGLE QUOTATION MARK
‚       .9      201A    8218    SINGLE LOW-9 QUOTATION MARK
‛       9'      201B    8219    SINGLE HIGH-REVERSED-9 QUOTATION MARK
“       "6      201C    8220    LEFT DOUBLE QUOTATION MARK
”       "9      201D    8221    RIGHT DOUBLE QUOTATION MARK
„       :9      201E    8222    DOUBLE LOW-9 QUOTATION MARK
‟       9"      201F    8223    DOUBLE HIGH-REVERSED-9 QUOTATION MARK
†       /-      2020    8224    DAGGER
‡       /=      2021    8225    DOUBLE DAGGER
‥       ..      2025    8229    TWO DOT LEADER
…       ,.      2026    8230    HORIZONTAL ELLIPSIS
‰       %0      2030    8240    PER MILLE SIGN
′       1'      2032    8242    PRIME
″       2'      2033    8243    DOUBLE PRIME
‴       3'      2034    8244    TRIPLE PRIME
‵       1"      2035    8245    REVERSED PRIME
‶       2"      2036    8246    REVERSED DOUBLE PRIME
‷       3"      2037    8247    REVERSED TRIPLE PRIME
‸       Ca      2038    8248    CARET
‹       <1      2039    8249    SINGLE LEFT-POINTING ANGLE QUOTATION MARK
›       >1      203A    8250    SINGLE RIGHT-POINTING ANGLE QUOTATION MARK
※       :X      203B    8251    REFERENCE MARK
‾       '-      203E    8254    OVERLINE
⁄       /f      2044    8260    FRACTION SLASH
⁰       0S      2070    8304    SUPERSCRIPT ZERO
⁴       4S      2074    8308    SUPERSCRIPT FOUR
⁵       5S      2075    8309    SUPERSCRIPT FIVE
⁶       6S      2076    8310    SUPERSCRIPT SIX
⁷       7S      2077    8311    SUPERSCRIPT SEVEN
⁸       8S      2078    8312    SUPERSCRIPT EIGHT
⁹       9S      2079    8313    SUPERSCRIPT NINE
⁺       +S      207A    8314    SUPERSCRIPT PLUS SIGN
⁻       -S      207B    8315    SUPERSCRIPT MINUS
⁼       =S      207C    8316    SUPERSCRIPT EQUALS SIGN
⁽       (S      207D    8317    SUPERSCRIPT LEFT PARENTHESIS
⁾       )S      207E    8318    SUPERSCRIPT RIGHT PARENTHESIS
ⁿ       nS      207F    8319    SUPERSCRIPT LATIN SMALL LETTER N `
₀       0s      2080    8320    SUBSCRIPT ZERO
₁       1s      2081    8321    SUBSCRIPT ONE
₂       2s      2082    8322    SUBSCRIPT TWO
₃       3s      2083    8323    SUBSCRIPT THREE
₄       4s      2084    8324    SUBSCRIPT FOUR
₅       5s      2085    8325    SUBSCRIPT FIVE
₆       6s      2086    8326    SUBSCRIPT SIX
₇       7s      2087    8327    SUBSCRIPT SEVEN
₈       8s      2088    8328    SUBSCRIPT EIGHT
```

₉	9s	2089	8329	SUBSCRIPT NINE
₊	+s	208A	8330	SUBSCRIPT PLUS SIGN
₋	-s	208B	8331	SUBSCRIPT MINUS
₌	=s	208C	8332	SUBSCRIPT EQUALS SIGN
₍	(s	208D	8333	SUBSCRIPT LEFT PARENTHESIS
₎	)s	208E	8334	SUBSCRIPT RIGHT PARENTHESIS
₤	Li	20A4	8356	LIRA SIGN
₧	Pt	20A7	8359	PESETA SIGN
₩	W=	20A9	8361	WON SIGN
€	Eu	20AC	8364	EURO SIGN
₽	=R	20BD	8381	ROUBLE SIGN
₽	=P	20BD	8381	ROUBLE SIGN
℃	oC	2103	8451	DEGREE CELSIUS
℅	co	2105	8453	CARE OF
℉	oF	2109	8457	DEGREE FAHRENHEIT
№	N0	2116	8470	NUMERO SIGN
℗	PO	2117	8471	SOUND RECORDING COPYRIGHT
℞	Rx	211E	8478	PRESCRIPTION TAKE
℠	SM	2120	8480	SERVICE MARK
™	TM	2122	8482	TRADE MARK SIGN
Ω	Om	2126	8486	OHM SIGN
Å	AO	212B	8491	ANGSTROM SIGN
⅓	13	2153	8531	VULGAR FRACTION ONE THIRD
⅔	23	2154	8532	VULGAR FRACTION TWO THIRDS
⅕	15	2155	8533	VULGAR FRACTION ONE FIFTH
⅖	25	2156	8534	VULGAR FRACTION TWO FIFTHS
⅗	35	2157	8535	VULGAR FRACTION THREE FIFTHS
⅘	45	2158	8536	VULGAR FRACTION FOUR FIFTHS
⅙	16	2159	8537	VULGAR FRACTION ONE SIXTH
⅚	56	215A	8538	VULGAR FRACTION FIVE SIXTHS
⅛	18	215B	8539	VULGAR FRACTION ONE EIGHTH
⅜	38	215C	8540	VULGAR FRACTION THREE EIGHTHS
⅝	58	215D	8541	VULGAR FRACTION FIVE EIGHTHS
⅞	78	215E	8542	VULGAR FRACTION SEVEN EIGHTHS
Ⅰ	1R	2160	8544	ROMAN NUMERAL ONE
Ⅱ	2R	2161	8545	ROMAN NUMERAL TWO
Ⅲ	3R	2162	8546	ROMAN NUMERAL THREE
Ⅳ	4R	2163	8547	ROMAN NUMERAL FOUR
Ⅴ	5R	2164	8548	ROMAN NUMERAL FIVE
Ⅵ	6R	2165	8549	ROMAN NUMERAL SIX
Ⅶ	7R	2166	8550	ROMAN NUMERAL SEVEN
Ⅷ	8R	2167	8551	ROMAN NUMERAL EIGHT
Ⅸ	9R	2168	8552	ROMAN NUMERAL NINE
Ⅹ	aR	2169	8553	ROMAN NUMERAL TEN
Ⅺ	bR	216A	8554	ROMAN NUMERAL ELEVEN
Ⅻ	cR	216B	8555	ROMAN NUMERAL TWELVE
ⅰ	1r	2170	8560	SMALL ROMAN NUMERAL ONE
ⅱ	2r	2171	8561	SMALL ROMAN NUMERAL TWO
ⅲ	3r	2172	8562	SMALL ROMAN NUMERAL THREE
ⅳ	4r	2173	8563	SMALL ROMAN NUMERAL FOUR
ⅴ	5r	2174	8564	SMALL ROMAN NUMERAL FIVE
ⅵ	6r	2175	8565	SMALL ROMAN NUMERAL SIX
ⅶ	7r	2176	8566	SMALL ROMAN NUMERAL SEVEN
ⅷ	8r	2177	8567	SMALL ROMAN NUMERAL EIGHT
ⅸ	9r	2178	8568	SMALL ROMAN NUMERAL NINE
ⅹ	ar	2179	8569	SMALL ROMAN NUMERAL TEN
ⅺ	br	217A	8570	SMALL ROMAN NUMERAL ELEVEN
ⅻ	cr	217B	8571	SMALL ROMAN NUMERAL TWELVE
←	<-	2190	8592	LEFTWARDS ARROW

↑	-!	2191	8593	UPWARDS ARROW
→	->	2192	8594	RIGHTWARDS ARROW
↓	-v	2193	8595	DOWNWARDS ARROW
↔	<>	2194	8596	LEFT RIGHT ARROW
↕	UD	2195	8597	UP DOWN ARROW
⇐	<=	21D0	8656	LEFTWARDS DOUBLE ARROW
⇒	=>	21D2	8658	RIGHTWARDS DOUBLE ARROW
⇔	==	21D4	8660	LEFT RIGHT DOUBLE ARROW
∀	FA	2200	8704	FOR ALL
∂	dP	2202	8706	PARTIAL DIFFERENTIAL
∃	TE	2203	8707	THERE EXISTS
∅	/0	2205	8709	EMPTY SET
∆	DE	2206	8710	INCREMENT
∇	NB	2207	8711	NABLA
∈	(-	2208	8712	ELEMENT OF
∋	-)	220B	8715	CONTAINS AS MEMBER
∏	*P	220F	8719	N-ARY PRODUCT `
∑	+Z	2211	8721	N-ARY SUMMATION `
−	-2	2212	8722	MINUS SIGN
∓	-+	2213	8723	MINUS-OR-PLUS SIGN
∗	*-	2217	8727	ASTERISK OPERATOR
∘	Ob	2218	8728	RING OPERATOR
∙	Sb	2219	8729	BULLET OPERATOR
√	RT	221A	8730	SQUARE ROOT
∝	0(	221D	8733	PROPORTIONAL TO
∞	00	221E	8734	INFINITY
∟	-L	221F	8735	RIGHT ANGLE
∠	-V	2220	8736	ANGLE
∥	PP	2225	8741	PARALLEL TO
∧	AN	2227	8743	LOGICAL AND
∨	OR	2228	8744	LOGICAL OR
∩	(U	2229	8745	INTERSECTION
∪	)U	222A	8746	UNION
∫	In	222B	8747	INTEGRAL
∬	DI	222C	8748	DOUBLE INTEGRAL
∮	Io	222E	8750	CONTOUR INTEGRAL
∴	.:	2234	8756	THEREFORE
∵	:.	2235	8757	BECAUSE
∶	:R	2236	8758	RATIO
∷	::	2237	8759	PROPORTION
∼	?1	223C	8764	TILDE OPERATOR
∾	CG	223E	8766	INVERTED LAZY S
≃	?-	2243	8771	ASYMPTOTICALLY EQUAL TO
≅	?=	2245	8773	APPROXIMATELY EQUAL TO
≈	?2	2248	8776	ALMOST EQUAL TO
≌	=?	224C	8780	ALL EQUAL TO
≓	HI	2253	8787	IMAGE OF OR APPROXIMATELY EQUAL TO
≠	!=	2260	8800	NOT EQUAL TO
≡	=3	2261	8801	IDENTICAL TO
≤	=<	2264	8804	LESS-THAN OR EQUAL TO
≥	>=	2265	8805	GREATER-THAN OR EQUAL TO
≪	<*	226A	8810	MUCH LESS-THAN
≫	*>	226B	8811	MUCH GREATER-THAN
≮	!<	226E	8814	NOT LESS-THAN
≯	!>	226F	8815	NOT GREATER-THAN
⊂	(C	2282	8834	SUBSET OF
⊃	)C	2283	8835	SUPERSET OF
⊆	(_	2286	8838	SUBSET OF OR EQUAL TO
⊇	)_	2287	8839	SUPERSET OF OR EQUAL TO
⊙	0.	2299	8857	CIRCLED DOT OPERATOR
⊚	02	229A	8858	CIRCLED RING OPERATOR

⊥	-T	22A5	8869	UP TACK
⋅	.P	22C5	8901	DOT OPERATOR
⋮	:3	22EE	8942	VERTICAL ELLIPSIS
⋯	.3	22EF	8943	MIDLINE HORIZONTAL ELLIPSIS
⌂	Eh	2302	8962	HOUSE
⌈	<7	2308	8968	LEFT CEILING
⌉	>7	2309	8969	RIGHT CEILING
⌊	7<	230A	8970	LEFT FLOOR
⌋	7>	230B	8971	RIGHT FLOOR
⌐	NI	2310	8976	REVERSED NOT SIGN
⌒	(A	2312	8978	ARC
⌕	TR	2315	8981	TELEPHONE RECORDER
⌠	Iu	2320	8992	TOP HALF INTEGRAL
⌡	Il	2321	8993	BOTTOM HALF INTEGRAL
〈	</	2329	9001	LEFT-POINTING ANGLE BRACKET
〉	/>	232A	9002	RIGHT-POINTING ANGLE BRACKET
␣	Vs	2423	9251	OPEN BOX
⑀	1h	2440	9280	OCR HOOK
⑁	3h	2441	9281	OCR CHAIR
⑂	2h	2442	9282	OCR FORK
⑃	4h	2443	9283	OCR INVERTED FORK
⑆	1j	2446	9286	OCR BRANCH BANK IDENTIFICATION
⑇	2j	2447	9287	OCR AMOUNT OF CHECK
⑈	3j	2448	9288	OCR DASH
⑉	4j	2449	9289	OCR CUSTOMER ACCOUNT NUMBER
⒈	1.	2488	9352	DIGIT ONE FULL STOP
⒉	2.	2489	9353	DIGIT TWO FULL STOP
⒊	3.	248A	9354	DIGIT THREE FULL STOP
⒋	4.	248B	9355	DIGIT FOUR FULL STOP
⒌	5.	248C	9356	DIGIT FIVE FULL STOP
⒍	6.	248D	9357	DIGIT SIX FULL STOP
⒎	7.	248E	9358	DIGIT SEVEN FULL STOP
⒏	8.	248F	9359	DIGIT EIGHT FULL STOP
⒐	9.	2490	9360	DIGIT NINE FULL STOP
─	hh	2500	9472	BOX DRAWINGS LIGHT HORIZONTAL
━	HH	2501	9473	BOX DRAWINGS HEAVY HORIZONTAL
│	vv	2502	9474	BOX DRAWINGS LIGHT VERTICAL
┃	VV	2503	9475	BOX DRAWINGS HEAVY VERTICAL
┄	3-	2504	9476	BOX DRAWINGS LIGHT TRIPLE DASH HORIZONTAL
┅	3_	2505	9477	BOX DRAWINGS HEAVY TRIPLE DASH HORIZONTAL
┆	3!	2506	9478	BOX DRAWINGS LIGHT TRIPLE DASH VERTICAL
┇	3/	2507	9479	BOX DRAWINGS HEAVY TRIPLE DASH VERTICAL
┈	4-	2508	9480	BOX DRAWINGS LIGHT QUADRUPLE DASH HORIZONTAL
┉	4_	2509	9481	BOX DRAWINGS HEAVY QUADRUPLE DASH HORIZONTAL
┊	4!	250A	9482	BOX DRAWINGS LIGHT QUADRUPLE DASH VERTICAL
┋	4/	250B	9483	BOX DRAWINGS HEAVY QUADRUPLE DASH VERTICAL
┌	dr	250C	9484	BOX DRAWINGS LIGHT DOWN AND RIGHT
┍	dR	250D	9485	BOX DRAWINGS DOWN LIGHT AND RIGHT HEAVY
┎	Dr	250E	9486	BOX DRAWINGS DOWN HEAVY AND RIGHT LIGHT
┏	DR	250F	9487	BOX DRAWINGS HEAVY DOWN AND RIGHT
┐	dl	2510	9488	BOX DRAWINGS LIGHT DOWN AND LEFT
┑	dL	2511	9489	BOX DRAWINGS DOWN LIGHT AND LEFT HEAVY
┒	Dl	2512	9490	BOX DRAWINGS DOWN HEAVY AND LEFT LIGHT
┓	LD	2513	9491	BOX DRAWINGS HEAVY DOWN AND LEFT
└	ur	2514	9492	BOX DRAWINGS LIGHT UP AND RIGHT
┕	uR	2515	9493	BOX DRAWINGS UP LIGHT AND RIGHT HEAVY
┖	Ur	2516	9494	BOX DRAWINGS UP HEAVY AND RIGHT LIGHT
┗	UR	2517	9495	BOX DRAWINGS HEAVY UP AND RIGHT
┘	ul	2518	9496	BOX DRAWINGS LIGHT UP AND LEFT
┙	uL	2519	9497	BOX DRAWINGS UP LIGHT AND LEFT HEAVY
┚	Ul	251A	9498	BOX DRAWINGS UP HEAVY AND LEFT LIGHT

```
┛       UL      251B    9499    BOX DRAWINGS HEAVY UP AND LEFT
├       vr      251C    9500    BOX DRAWINGS LIGHT VERTICAL AND RIGHT
┝       vR      251D    9501    BOX DRAWINGS VERTICAL LIGHT AND RIGHT HEAVY
┠       Vr      2520    9504    BOX DRAWINGS VERTICAL HEAVY AND RIGHT LIGHT
┣       VR      2523    9507    BOX DRAWINGS HEAVY VERTICAL AND RIGHT
┤       vl      2524    9508    BOX DRAWINGS LIGHT VERTICAL AND LEFT
┥       vL      2525    9509    BOX DRAWINGS VERTICAL LIGHT AND LEFT HEAVY
┨       Vl      2528    9512    BOX DRAWINGS VERTICAL HEAVY AND LEFT LIGHT
┫       VL      252B    9515    BOX DRAWINGS HEAVY VERTICAL AND LEFT
┬       dh      252C    9516    BOX DRAWINGS LIGHT DOWN AND HORIZONTAL
┯       dH      252F    9519    BOX DRAWINGS DOWN LIGHT AND HORIZONTAL HEAVY
┰       Dh      2530    9520    BOX DRAWINGS DOWN HEAVY AND HORIZONTAL LIGHT
┳       DH      2533    9523    BOX DRAWINGS HEAVY DOWN AND HORIZONTAL
┴       uh      2534    9524    BOX DRAWINGS LIGHT UP AND HORIZONTAL
┷       uH      2537    9527    BOX DRAWINGS UP LIGHT AND HORIZONTAL HEAVY
┸       Uh      2538    9528    BOX DRAWINGS UP HEAVY AND HORIZONTAL LIGHT
┻       UH      253B    9531    BOX DRAWINGS HEAVY UP AND HORIZONTAL
┼       vh      253C    9532    BOX DRAWINGS LIGHT VERTICAL AND HORIZONTAL
┿       vH      253F    9535    BOX DRAWINGS VERTICAL LIGHT AND HORIZONTAL HEAVY
╂       Vh      2542    9538    BOX DRAWINGS VERTICAL HEAVY AND HORIZONTAL LIGHT
╋       VH      254B    9547    BOX DRAWINGS HEAVY VERTICAL AND HORIZONTAL
╱       FD      2571    9585    BOX DRAWINGS LIGHT DIAGONAL UPPER RIGHT TO LOWER LEFT
╲       BD      2572    9586    BOX DRAWINGS LIGHT DIAGONAL UPPER LEFT TO LOWER RIGHT
▀       TB      2580    9600    UPPER HALF BLOCK
▄       LB      2584    9604    LOWER HALF BLOCK
█       FB      2588    9608    FULL BLOCK
▌       lB      258C    9612    LEFT HALF BLOCK
▐       RB      2590    9616    RIGHT HALF BLOCK
░       .S      2591    9617    LIGHT SHADE
▒       :S      2592    9618    MEDIUM SHADE
▓       ?S      2593    9619    DARK SHADE
■       fS      25A0    9632    BLACK SQUARE
□       OS      25A1    9633    WHITE SQUARE
▢       RO      25A2    9634    WHITE SQUARE WITH ROUNDED CORNERS
▣       Rr      25A3    9635    WHITE SQUARE CONTAINING BLACK SMALL SQUARE
▤       RF      25A4    9636    SQUARE WITH HORIZONTAL FILL
▥       RY      25A5    9637    SQUARE WITH VERTICAL FILL
▦       RH      25A6    9638    SQUARE WITH ORTHOGONAL CROSSHATCH FILL
▧       RZ      25A7    9639    SQUARE WITH UPPER LEFT TO LOWER RIGHT FILL
▨       RK      25A8    9640    SQUARE WITH UPPER RIGHT TO LOWER LEFT FILL
▩       RX      25A9    9641    SQUARE WITH DIAGONAL CROSSHATCH FILL
▪       sB      25AA    9642    BLACK SMALL SQUARE
▬       SR      25AC    9644    BLACK RECTANGLE
▭       Or      25AD    9645    WHITE RECTANGLE
▲       UT      25B2    9650    BLACK UP-POINTING TRIANGLE
△       uT      25B3    9651    WHITE UP-POINTING TRIANGLE
▶       PR      25B6    9654    BLACK RIGHT-POINTING TRIANGLE
▷       Tr      25B7    9655    WHITE RIGHT-POINTING TRIANGLE
▼       Dt      25BC    9660    BLACK DOWN-POINTING TRIANGLE
▽       dT      25BD    9661    WHITE DOWN-POINTING TRIANGLE
◀       PL      25C0    9664    BLACK LEFT-POINTING TRIANGLE
◁       Tl      25C1    9665    WHITE LEFT-POINTING TRIANGLE
◆       Db      25C6    9670    BLACK DIAMOND
◇       Dw      25C7    9671    WHITE DIAMOND
◊       LZ      25CA    9674    LOZENGE
○       0m      25CB    9675    WHITE CIRCLE
◎       0o      25CE    9678    BULLSEYE
●       0M      25CF    9679    BLACK CIRCLE
◐       0L      25D0    9680    CIRCLE WITH LEFT HALF BLACK
◑       0R      25D1    9681    CIRCLE WITH RIGHT HALF BLACK
◘       Sn      25D8    9688    INVERSE BULLET
◙       Ic      25D9    9689    INVERSE WHITE CIRCLE
```

◢	Fd	25E2	9698	BLACK LOWER RIGHT TRIANGLE
◣	Bd	25E3	9699	BLACK LOWER LEFT TRIANGLE
★	*2	2605	9733	BLACK STAR
☆	*1	2606	9734	WHITE STAR
☜	<H	261C	9756	WHITE LEFT POINTING INDEX
☞	>H	261E	9758	WHITE RIGHT POINTING INDEX
☺	0u	263A	9786	WHITE SMILING FACE
☻	0U	263B	9787	BLACK SMILING FACE
☼	SU	263C	9788	WHITE SUN WITH RAYS
♀	Fm	2640	9792	FEMALE SIGN
♂	Ml	2642	9794	MALE SIGN
♠	cS	2660	9824	BLACK SPADE SUIT
♡	cH	2661	9825	WHITE HEART SUIT
♢	cD	2662	9826	WHITE DIAMOND SUIT
♣	cC	2663	9827	BLACK CLUB SUIT
♩	Md	2669	9833	QUARTER NOTE `
♪	M8	266A	9834	EIGHTH NOTE `
♫	M2	266B	9835	BEAMED EIGHTH NOTES
♭	Mb	266D	9837	MUSIC FLAT SIGN
♮	Mx	266E	9838	MUSIC NATURAL SIGN
♯	MX	266F	9839	MUSIC SHARP SIGN
✓	OK	2713	10003	CHECK MARK
✗	XX	2717	10007	BALLOT X
✠	-X	2720	10016	MALTESE CROSS
	IS	3000	12288	IDEOGRAPHIC SPACE
、	,_	3001	12289	IDEOGRAPHIC COMMA
。	._	3002	12290	IDEOGRAPHIC FULL STOP
〃	+"	3003	12291	DITTO MARK
〄	+_	3004	12292	JAPANESE INDUSTRIAL STANDARD SYMBOL
々	*_	3005	12293	IDEOGRAPHIC ITERATION MARK
〆	;_	3006	12294	IDEOGRAPHIC CLOSING MARK
〇	0_	3007	12295	IDEOGRAPHIC NUMBER ZERO
《	<+	300A	12298	LEFT DOUBLE ANGLE BRACKET
》	>+	300B	12299	RIGHT DOUBLE ANGLE BRACKET
「	<'	300C	12300	LEFT CORNER BRACKET
」	>'	300D	12301	RIGHT CORNER BRACKET
『	<"	300E	12302	LEFT WHITE CORNER BRACKET
』	>"	300F	12303	RIGHT WHITE CORNER BRACKET
【	("	3010	12304	LEFT BLACK LENTICULAR BRACKET
】	)"	3011	12305	RIGHT BLACK LENTICULAR BRACKET
〒	=T	3012	12306	POSTAL MARK
〓	=_	3013	12307	GETA MARK
〔	('	3014	12308	LEFT TORTOISE SHELL BRACKET
〕	)'	3015	12309	RIGHT TORTOISE SHELL BRACKET
〖	(I	3016	12310	LEFT WHITE LENTICULAR BRACKET
〗	)I	3017	12311	RIGHT WHITE LENTICULAR BRACKET
〜	-?	301C	12316	WAVE DASH
ぁ	A5	3041	12353	HIRAGANA LETTER SMALL A
あ	a5	3042	12354	HIRAGANA LETTER A
ぃ	I5	3043	12355	HIRAGANA LETTER SMALL I
い	i5	3044	12356	HIRAGANA LETTER I
ぅ	U5	3045	12357	HIRAGANA LETTER SMALL U
う	u5	3046	12358	HIRAGANA LETTER U
ぇ	E5	3047	12359	HIRAGANA LETTER SMALL E
え	e5	3048	12360	HIRAGANA LETTER E
ぉ	O5	3049	12361	HIRAGANA LETTER SMALL O
お	o5	304A	12362	HIRAGANA LETTER O
か	ka	304B	12363	HIRAGANA LETTER KA
が	ga	304C	12364	HIRAGANA LETTER GA

き	ki	304D	12365	HIRAGANA LETTER KI
ぎ	gi	304E	12366	HIRAGANA LETTER GI
く	ku	304F	12367	HIRAGANA LETTER KU
ぐ	gu	3050	12368	HIRAGANA LETTER GU
け	ke	3051	12369	HIRAGANA LETTER KE
げ	ge	3052	12370	HIRAGANA LETTER GE
こ	ko	3053	12371	HIRAGANA LETTER KO
ご	go	3054	12372	HIRAGANA LETTER GO
さ	sa	3055	12373	HIRAGANA LETTER SA
ざ	za	3056	12374	HIRAGANA LETTER ZA
し	si	3057	12375	HIRAGANA LETTER SI
じ	zi	3058	12376	HIRAGANA LETTER ZI
す	su	3059	12377	HIRAGANA LETTER SU
ず	zu	305A	12378	HIRAGANA LETTER ZU
せ	se	305B	12379	HIRAGANA LETTER SE
ぜ	ze	305C	12380	HIRAGANA LETTER ZE
そ	so	305D	12381	HIRAGANA LETTER SO
ぞ	zo	305E	12382	HIRAGANA LETTER ZO
た	ta	305F	12383	HIRAGANA LETTER TA
だ	da	3060	12384	HIRAGANA LETTER DA
ち	ti	3061	12385	HIRAGANA LETTER TI
ぢ	di	3062	12386	HIRAGANA LETTER DI
っ	tU	3063	12387	HIRAGANA LETTER SMALL TU
つ	tu	3064	12388	HIRAGANA LETTER TU
づ	du	3065	12389	HIRAGANA LETTER DU
て	te	3066	12390	HIRAGANA LETTER TE
で	de	3067	12391	HIRAGANA LETTER DE
と	to	3068	12392	HIRAGANA LETTER TO
ど	do	3069	12393	HIRAGANA LETTER DO
な	na	306A	12394	HIRAGANA LETTER NA
に	ni	306B	12395	HIRAGANA LETTER NI
ぬ	nu	306C	12396	HIRAGANA LETTER NU
ね	ne	306D	12397	HIRAGANA LETTER NE
の	no	306E	12398	HIRAGANA LETTER NO
は	ha	306F	12399	HIRAGANA LETTER HA
ば	ba	3070	12400	HIRAGANA LETTER BA
ぱ	pa	3071	12401	HIRAGANA LETTER PA
ひ	hi	3072	12402	HIRAGANA LETTER HI
び	bi	3073	12403	HIRAGANA LETTER BI
ぴ	pi	3074	12404	HIRAGANA LETTER PI
ふ	hu	3075	12405	HIRAGANA LETTER HU
ぶ	bu	3076	12406	HIRAGANA LETTER BU
ぷ	pu	3077	12407	HIRAGANA LETTER PU
へ	he	3078	12408	HIRAGANA LETTER HE
べ	be	3079	12409	HIRAGANA LETTER BE
ぺ	pe	307A	12410	HIRAGANA LETTER PE
ほ	ho	307B	12411	HIRAGANA LETTER HO
ぼ	bo	307C	12412	HIRAGANA LETTER BO
ぽ	po	307D	12413	HIRAGANA LETTER PO
ま	ma	307E	12414	HIRAGANA LETTER MA
み	mi	307F	12415	HIRAGANA LETTER MI
む	mu	3080	12416	HIRAGANA LETTER MU
め	me	3081	12417	HIRAGANA LETTER ME
も	mo	3082	12418	HIRAGANA LETTER MO
ゃ	yA	3083	12419	HIRAGANA LETTER SMALL YA
や	ya	3084	12420	HIRAGANA LETTER YA

ゅ	yU	3085	12421	HIRAGANA LETTER SMALL YU
ゆ	yu	3086	12422	HIRAGANA LETTER YU
ょ	yO	3087	12423	HIRAGANA LETTER SMALL YO
よ	yo	3088	12424	HIRAGANA LETTER YO
ら	ra	3089	12425	HIRAGANA LETTER RA
り	ri	308A	12426	HIRAGANA LETTER RI
る	ru	308B	12427	HIRAGANA LETTER RU
れ	re	308C	12428	HIRAGANA LETTER RE
ろ	ro	308D	12429	HIRAGANA LETTER RO
ゎ	wA	308E	12430	HIRAGANA LETTER SMALL WA
わ	wa	308F	12431	HIRAGANA LETTER WA
ゐ	wi	3090	12432	HIRAGANA LETTER WI
ゑ	we	3091	12433	HIRAGANA LETTER WE
を	wo	3092	12434	HIRAGANA LETTER WO
ん	n5	3093	12435	HIRAGANA LETTER N `
ゔ	vu	3094	12436	HIRAGANA LETTER VU
゛	"5	309B	12443	KATAKANA-HIRAGANA VOICED SOUND MARK
゜	05	309C	12444	KATAKANA-HIRAGANA SEMI-VOICED SOUND MARK
ゝ	*5	309D	12445	HIRAGANA ITERATION MARK
ゞ	+5	309E	12446	HIRAGANA VOICED ITERATION MARK
ァ	a6	30A1	12449	KATAKANA LETTER SMALL A
ア	A6	30A2	12450	KATAKANA LETTER A
ィ	i6	30A3	12451	KATAKANA LETTER SMALL I
イ	I6	30A4	12452	KATAKANA LETTER I
ゥ	u6	30A5	12453	KATAKANA LETTER SMALL U
ウ	U6	30A6	12454	KATAKANA LETTER U
ェ	e6	30A7	12455	KATAKANA LETTER SMALL E
エ	E6	30A8	12456	KATAKANA LETTER E
ォ	o6	30A9	12457	KATAKANA LETTER SMALL O
オ	O6	30AA	12458	KATAKANA LETTER O
カ	Ka	30AB	12459	KATAKANA LETTER KA
ガ	Ga	30AC	12460	KATAKANA LETTER GA
キ	Ki	30AD	12461	KATAKANA LETTER KI
ギ	Gi	30AE	12462	KATAKANA LETTER GI
ク	Ku	30AF	12463	KATAKANA LETTER KU
グ	Gu	30B0	12464	KATAKANA LETTER GU
ケ	Ke	30B1	12465	KATAKANA LETTER KE
ゲ	Ge	30B2	12466	KATAKANA LETTER GE
コ	Ko	30B3	12467	KATAKANA LETTER KO
ゴ	Go	30B4	12468	KATAKANA LETTER GO
サ	Sa	30B5	12469	KATAKANA LETTER SA
ザ	Za	30B6	12470	KATAKANA LETTER ZA
シ	Si	30B7	12471	KATAKANA LETTER SI
ジ	Zi	30B8	12472	KATAKANA LETTER ZI
ス	Su	30B9	12473	KATAKANA LETTER SU
ズ	Zu	30BA	12474	KATAKANA LETTER ZU
セ	Se	30BB	12475	KATAKANA LETTER SE
ゼ	Ze	30BC	12476	KATAKANA LETTER ZE
ソ	So	30BD	12477	KATAKANA LETTER SO
ゾ	Zo	30BE	12478	KATAKANA LETTER ZO
タ	Ta	30BF	12479	KATAKANA LETTER TA
ダ	Da	30C0	12480	KATAKANA LETTER DA
チ	Ti	30C1	12481	KATAKANA LETTER TI
ヂ	Di	30C2	12482	KATAKANA LETTER DI
ッ	TU	30C3	12483	KATAKANA LETTER SMALL TU
ツ	Tu	30C4	12484	KATAKANA LETTER TU

ヅ	Du	30C5	12485	KATAKANA LETTER DU
テ	Te	30C6	12486	KATAKANA LETTER TE
デ	De	30C7	12487	KATAKANA LETTER DE
ト	To	30C8	12488	KATAKANA LETTER TO
ド	Do	30C9	12489	KATAKANA LETTER DO
ナ	Na	30CA	12490	KATAKANA LETTER NA
ニ	Ni	30CB	12491	KATAKANA LETTER NI
ヌ	Nu	30CC	12492	KATAKANA LETTER NU
ネ	Ne	30CD	12493	KATAKANA LETTER NE
ノ	No	30CE	12494	KATAKANA LETTER NO
ハ	Ha	30CF	12495	KATAKANA LETTER HA
バ	Ba	30D0	12496	KATAKANA LETTER BA
パ	Pa	30D1	12497	KATAKANA LETTER PA
ヒ	Hi	30D2	12498	KATAKANA LETTER HI
ビ	Bi	30D3	12499	KATAKANA LETTER BI
ピ	Pi	30D4	12500	KATAKANA LETTER PI
フ	Hu	30D5	12501	KATAKANA LETTER HU
ブ	Bu	30D6	12502	KATAKANA LETTER BU
プ	Pu	30D7	12503	KATAKANA LETTER PU
ヘ	He	30D8	12504	KATAKANA LETTER HE
ベ	Be	30D9	12505	KATAKANA LETTER BE
ペ	Pe	30DA	12506	KATAKANA LETTER PE
ホ	Ho	30DB	12507	KATAKANA LETTER HO
ボ	Bo	30DC	12508	KATAKANA LETTER BO
ポ	Po	30DD	12509	KATAKANA LETTER PO
マ	Ma	30DE	12510	KATAKANA LETTER MA
ミ	Mi	30DF	12511	KATAKANA LETTER MI
ム	Mu	30E0	12512	KATAKANA LETTER MU
メ	Me	30E1	12513	KATAKANA LETTER ME
モ	Mo	30E2	12514	KATAKANA LETTER MO
ャ	YA	30E3	12515	KATAKANA LETTER SMALL YA
ヤ	Ya	30E4	12516	KATAKANA LETTER YA
ュ	YU	30E5	12517	KATAKANA LETTER SMALL YU
ユ	Yu	30E6	12518	KATAKANA LETTER YU
ョ	YO	30E7	12519	KATAKANA LETTER SMALL YO
ヨ	Yo	30E8	12520	KATAKANA LETTER YO
ラ	Ra	30E9	12521	KATAKANA LETTER RA
リ	Ri	30EA	12522	KATAKANA LETTER RI
ル	Ru	30EB	12523	KATAKANA LETTER RU
レ	Re	30EC	12524	KATAKANA LETTER RE
ロ	Ro	30ED	12525	KATAKANA LETTER RO
ヮ	WA	30EE	12526	KATAKANA LETTER SMALL WA
ワ	Wa	30EF	12527	KATAKANA LETTER WA
ヰ	Wi	30F0	12528	KATAKANA LETTER WI
ヱ	We	30F1	12529	KATAKANA LETTER WE
ヲ	Wo	30F2	12530	KATAKANA LETTER WO
ン	N6	30F3	12531	KATAKANA LETTER N `
ヴ	Vu	30F4	12532	KATAKANA LETTER VU
ヵ	KA	30F5	12533	KATAKANA LETTER SMALL KA
ヶ	KE	30F6	12534	KATAKANA LETTER SMALL KE
ヷ	Va	30F7	12535	KATAKANA LETTER VA
ヸ	Vi	30F8	12536	KATAKANA LETTER VI
ヹ	Ve	30F9	12537	KATAKANA LETTER VE
ヺ	Vo	30FA	12538	KATAKANA LETTER VO
・	.6	30FB	12539	KATAKANA MIDDLE DOT
ー	-6	30FC	12540	KATAKANA-HIRAGANA PROLONGED SOUND MARK

```
ヽ       *6      30FD    12541   KATAKANA ITERATION MARK
ヾ       +6      30FE    12542   KATAKANA VOICED ITERATION MARK
ㄅ       b4      3105    12549   BOPOMOFO LETTER B
ㄆ       p4      3106    12550   BOPOMOFO LETTER P
ㄇ       m4      3107    12551   BOPOMOFO LETTER M
ㄈ       f4      3108    12552   BOPOMOFO LETTER F
ㄉ       d4      3109    12553   BOPOMOFO LETTER D
ㄊ       t4      310A    12554   BOPOMOFO LETTER T
ㄋ       n4      310B    12555   BOPOMOFO LETTER N `
ㄌ       l4      310C    12556   BOPOMOFO LETTER L
ㄍ       g4      310D    12557   BOPOMOFO LETTER G
ㄎ       k4      310E    12558   BOPOMOFO LETTER K
ㄏ       h4      310F    12559   BOPOMOFO LETTER H
ㄐ       j4      3110    12560   BOPOMOFO LETTER J
ㄑ       q4      3111    12561   BOPOMOFO LETTER Q
ㄒ       x4      3112    12562   BOPOMOFO LETTER X
ㄓ       zh      3113    12563   BOPOMOFO LETTER ZH
ㄔ       ch      3114    12564   BOPOMOFO LETTER CH
ㄕ       sh      3115    12565   BOPOMOFO LETTER SH
ㄖ       r4      3116    12566   BOPOMOFO LETTER R
ㄗ       z4      3117    12567   BOPOMOFO LETTER Z
ㄘ       c4      3118    12568   BOPOMOFO LETTER C
ㄙ       s4      3119    12569   BOPOMOFO LETTER S
ㄚ       a4      311A    12570   BOPOMOFO LETTER A
ㄛ       o4      311B    12571   BOPOMOFO LETTER O
ㄜ       e4      311C    12572   BOPOMOFO LETTER E
ㄞ       ai      311E    12574   BOPOMOFO LETTER AI
ㄟ       ei      311F    12575   BOPOMOFO LETTER EI
ㄠ       au      3120    12576   BOPOMOFO LETTER AU
ㄡ       ou      3121    12577   BOPOMOFO LETTER OU
ㄢ       an      3122    12578   BOPOMOFO LETTER AN
ㄣ       en      3123    12579   BOPOMOFO LETTER EN
ㄤ       aN      3124    12580   BOPOMOFO LETTER ANG
ㄥ       eN      3125    12581   BOPOMOFO LETTER ENG
ㄦ       er      3126    12582   BOPOMOFO LETTER ER
ㄧ       i4      3127    12583   BOPOMOFO LETTER I
ㄨ       u4      3128    12584   BOPOMOFO LETTER U
ㄩ       iu      3129    12585   BOPOMOFO LETTER IU
ㄪ       v4      312A    12586   BOPOMOFO LETTER V
ㄫ       nG      312B    12587   BOPOMOFO LETTER NG
ㄬ       gn      312C    12588   BOPOMOFO LETTER GN
㈠       1c      3220    12832   PARENTHESIZED IDEOGRAPH ONE
㈡       2c      3221    12833   PARENTHESIZED IDEOGRAPH TWO
㈢       3c      3222    12834   PARENTHESIZED IDEOGRAPH THREE
㈣       4c      3223    12835   PARENTHESIZED IDEOGRAPH FOUR
㈤       5c      3224    12836   PARENTHESIZED IDEOGRAPH FIVE
㈥       6c      3225    12837   PARENTHESIZED IDEOGRAPH SIX
㈦       7c      3226    12838   PARENTHESIZED IDEOGRAPH SEVEN
㈧       8c      3227    12839   PARENTHESIZED IDEOGRAPH EIGHT
㈨       9c      3228    12840   PARENTHESIZED IDEOGRAPH NINE
```

```
ﬀ       ff      FB00    64256   LATIN SMALL LIGATURE FF
ﬁ       fi      FB01    64257   LATIN SMALL LIGATURE FI
ﬂ       fl      FB02    64258   LATIN SMALL LIGATURE FL
ﬅ       ft      FB05    64261   LATIN SMALL LIGATURE LONG S T
ﬆ       st      FB06    64262   LATIN SMALL LIGATURE ST

 vim:tw=78:ts=8:ft=help:norl:
*mbyte.txt*     For Vim version 8.0.  Last change: 2016 Jul 21


                  VIM REFERENCE MANUAL    by Bram Moolenaar et al.


Multi-byte support                              *multibyte* *multi-byte*
                                                *Chinese* *Japanese* *Korean*
This is about editing text in languages which have many characters that can
not be represented using one byte (one octet).  Examples are Chinese, Japanese
and Korean.  Unicode is also covered here.

For an introduction to the most common features, see |usr_45.txt| in the user
manual.
For changing the language of messages and menus see |mlang.txt|.

{not available when compiled without the |+multi_byte| feature}


1.  Getting started                     |mbyte-first|
2.  Locale                              |mbyte-locale|
3.  Encoding                            |mbyte-encoding|
4.  Using a terminal                    |mbyte-terminal|
5.  Fonts on X11                        |mbyte-fonts-X11|
6.  Fonts on MS-Windows                 |mbyte-fonts-MSwin|
7.  Input on X11                        |mbyte-XIM|
8.  Input on MS-Windows                 |mbyte-IME|
9.  Input with a keymap                 |mbyte-keymap|
10. Using UTF-8                         |mbyte-utf8|
11. Overview of options                 |mbyte-options|

NOTE: This file contains UTF-8 characters.  These may show up as strange
characters or boxes when using another encoding.

==============================================================================
1. Getting started                                      *mbyte-first*

This is a summary of the multibyte features in Vim.  If you are lucky it works
as described and you can start using Vim without much trouble.  If something
doesn't work you will have to read the rest.  Don't be surprised if it takes
quite a bit of work and experimenting to make Vim use all the multi-byte
features.  Unfortunately, every system has its own way to deal with multibyte
languages and it is quite complicated.


COMPILING

If you already have a compiled Vim program, check if the |+multi_byte| feature
is included.  The |:version| command can be used for this.

If +multi_byte is not included, you should compile Vim with "normal", "big" or
"huge" features.  You can further tune what features are included.  See the
INSTALL files in the source directory.
```

LOCALE

First of all, you must make sure your current locale is set correctly. If your system has been installed to use the language, it probably works right away. If not, you can often make it work by setting the $LANG environment variable in your shell: >

```
        setenv LANG ja_JP.EUC
```

Unfortunately, the name of the locale depends on your system. Japanese might also be called "ja_JP.EUCjp" or just "ja". To see what is currently used: >

```
        :language
```

To change the locale inside Vim use: >

```
        :language ja_JP.EUC
```

Vim will give an error message if this doesn't work. This is a good way to experiment and find the locale name you want to use. But it's always better to set the locale in the shell, so that it is used right from the start.

See |mbyte-locale| for details.

ENCODING

If your locale works properly, Vim will try to set the 'encoding' option accordingly. If this doesn't work you can overrule its value: >

```
        :set encoding=utf-8
```

See |encoding-values| for a list of acceptable values.

The result is that all the text that is used inside Vim will be in this encoding. Not only the text in the buffers, but also in registers, variables, etc. This also means that changing the value of 'encoding' makes the existing text invalid! The text doesn't change, but it will be displayed wrong.

You can edit files in another encoding than what 'encoding' is set to. Vim will convert the file when you read it and convert it back when you write it. See 'fileencoding', 'fileencodings' and |++enc|.

DISPLAY AND FONTS

If you are working in a terminal (emulator) you must make sure it accepts the same encoding as which Vim is working with. If this is not the case, you can use the 'termencoding' option to make Vim convert text automatically.

For the GUI you must select fonts that work with the current 'encoding'. This is the difficult part. It depends on the system you are using, the locale and a few other things. See the chapters on fonts: |mbyte-fonts-X11| for X-Windows and |mbyte-fonts-MSwin| for MS-Windows.

For GTK+ 2, you can skip most of this section. The option 'guifontset' does no longer exist. You only need to set 'guifont' and everything should "just work". If your system comes with Xft2 and fontconfig and the current font does not contain a certain glyph, a different font will be used automatically if available. The 'guifontwide' option is still supported but usually you do not need to set it. It is only necessary if the automatic font selection does not suit your needs.

```
For X11 you can set the 'guifontset' option to a list of fonts that together
cover the characters that are used.  Example for Korean: >

        :set guifontset=k12,r12

Alternatively, you can set 'guifont' and 'guifontwide'.  'guifont' is used for
the single-width characters, 'guifontwide' for the double-width characters.
Thus the 'guifontwide' font must be exactly twice as wide as 'guifont'.
Example for UTF-8: >

        :set guifont=-misc-fixed-medium-r-normal-*-18-120-100-100-c-90-iso10646-1
        :set guifontwide=-misc-fixed-medium-r-normal-*-18-120-100-100-c-180-iso10646-1

You can also set 'guifont' alone, Vim will try to find a matching
'guifontwide' for you.


INPUT

There are several ways to enter multi-byte characters:
- For X11 XIM can be used.  See |XIM|.
- For MS-Windows IME can be used.  See |IME|.
- For all systems keymaps can be used.  See |mbyte-keymap|.

The options 'iminsert', 'imsearch' and 'imcmdline' can be used to chose
the different input methods or disable them temporarily.

==============================================================================
2.  Locale                                              *mbyte-locale*

The easiest setup is when your whole system uses the locale you want to work
in.  But it's also possible to set the locale for one shell you are working
in, or just use a certain locale inside Vim.


WHAT IS A LOCALE?                                       *locale*

There are many of languages in the world.  And there are different cultures
and environments at least as much as the number of languages.   A linguistic
environment corresponding to an area is called "locale".  This includes
information about the used language, the charset, collating order for sorting,
date format, currency format and so on.  For Vim only the language and charset
really matter.

You can only use a locale if your system has support for it.  Some systems
have only a few locales, especially in the USA.  The language which you want
to use may not be on your system.  In that case you might be able to install
it as an extra package.  Check your system documentation for how to do that.

The location in which the locales are installed varies from system to system.
For example, "/usr/share/locale" or "/usr/lib/locale".  See your system's
setlocale() man page.

Looking in these directories will show you the exact name of each locale.
Mostly upper/lowercase matters, thus "ja_JP.EUC" and "ja_jp.euc" are
different.  Some systems have a locale.alias file, which allows translation
from a short name like "nl" to the full name "nl_NL.ISO_8859-1".

Note that X-windows has its own locale stuff.  And unfortunately uses locale
names different from what is used elsewhere.  This is confusing!  For Vim it
matters what the setlocale() function uses, which is generally NOT the
```

X-windows stuff. You might have to do some experiments to find out what really works.

locale-name

The (simplified) format of |locale| name is:

```
        language
or      language_territory
or      language_territory.codeset
```

Territory means the country (or part of it), codeset means the |charset|. For example, the locale name "ja_JP.eucJP" means:

```
        ja      the language is Japanese
        JP      the country is Japan
        eucJP   the codeset is EUC-JP
```

But it also could be "ja", "ja_JP.EUC", "ja_JP.ujis", etc. And unfortunately, the locale name for a specific language, territory and codeset is not unified and depends on your system.

Examples of locale name:

charset	language	locale name ~
GB2312	Chinese (simplified)	zh_CN.EUC, zh_CN.GB2312
Big5	Chinese (traditional)	zh_TW.BIG5, zh_TW.Big5
CNS-11643	Chinese (traditional)	zh_TW
EUC-JP	Japanese	ja, ja_JP.EUC, ja_JP.ujis, ja_JP.eucJP
Shift_JIS	Japanese	ja_JP.SJIS, ja_JP.Shift_JIS
EUC-KR	Korean	ko, ko_KR.EUC

USING A LOCALE

To start using a locale for the whole system, see the documentation of your system. Mostly you need to set it in a configuration file in "/etc".

To use a locale in a shell, set the $LANG environment value. When you want to use Korean and the |locale| name is "ko", do this:

```
    sh:    export LANG=ko
    csh:   setenv LANG ko
```

You can put this in your ~/.profile or ~/.cshrc file to always use it.

To use a locale in Vim only, use the |:language| command: >

```
        :language ko
```

Put this in your ~/.vimrc file to use it always.

Or specify $LANG when starting Vim:

```
   sh:    LANG=ko vim {vim-arguments}
   csh:   env LANG=ko vim {vim-arguments}
```

You could make a small shell script for this.

==

3. Encoding *mbyte-encoding*

Vim uses the 'encoding' option to specify how characters are identified and encoded when they are used inside Vim. This applies to all the places where text is used, including buffers (files loaded into memory), registers and variables.

charset *codeset*

Charset is another name for encoding. There are subtle differences, but these don't matter when using Vim. "codeset" is another similar name.

Each character is encoded as one or more bytes. When all characters are encoded with one byte, we call this a single-byte encoding. The most often used one is called "latin1". This limits the number of characters to 256. Some of these are control characters, thus even fewer can be used for text.

When some characters use two or more bytes, we call this a multi-byte encoding. This allows using much more than 256 characters, which is required for most East Asian languages.

Most multi-byte encodings use one byte for the first 127 characters. These are equal to ASCII, which makes it easy to exchange plain-ASCII text, no matter what language is used. Thus you might see the right text even when the encoding was set wrong.

encoding-names

Vim can use many different character encodings. There are three major groups:

1	8bit	Single-byte encodings, 256 different characters. Mostly used in USA and Europe. Example: ISO-8859-1 (Latin1). All characters occupy one screen cell only.
2	2byte	Double-byte encodings, over 10000 different characters. Mostly used in Asian countries. Example: euc-kr (Korean) The number of screen cells is equal to the number of bytes (except for euc-jp when the first byte is 0x8e).
u	Unicode	Universal encoding, can replace all others. ISO 10646. Millions of different characters. Example: UTF-8. The relation between bytes and screen cells is complex.

Other encodings cannot be used by Vim internally. But files in other encodings can be edited by using conversion, see 'fileencoding'.
Note that all encodings must use ASCII for the characters up to 128 (except when compiled for EBCDIC).

Supported 'encoding' values are: *encoding-values*

1	latin1	8-bit characters (ISO 8859-1, also used for cp1252)
1	iso-8859-n	ISO_8859 variant (n = 2 to 15)
1	koi8-r	Russian
1	koi8-u	Ukrainian
1	macroman	MacRoman (Macintosh encoding)
1	8bit-{name}	any 8-bit encoding (Vim specific name)
1	cp437	similar to iso-8859-1
1	cp737	similar to iso-8859-7
1	cp775	Baltic
1	cp850	similar to iso-8859-4
1	cp852	similar to iso-8859-1
1	cp855	similar to iso-8859-2
1	cp857	similar to iso-8859-5
1	cp860	similar to iso-8859-9
1	cp861	similar to iso-8859-1
1	cp862	similar to iso-8859-1
1	cp863	similar to iso-8859-8
1	cp865	similar to iso-8859-1
1	cp866	similar to iso-8859-5
1	cp869	similar to iso-8859-7
1	cp874	Thai

```
1   cp1250      Czech, Polish, etc.
1   cp1251      Cyrillic
1   cp1253      Greek
1   cp1254      Turkish
1   cp1255      Hebrew
1   cp1256      Arabic
1   cp1257      Baltic
1   cp1258      Vietnamese
1   cp{number}  MS-Windows: any installed single-byte codepage
2   cp932       Japanese (Windows only)
2   euc-jp      Japanese (Unix only)
2   sjis        Japanese (Unix only)
2   cp949       Korean (Unix and Windows)
2   euc-kr      Korean (Unix only)
2   cp936       simplified Chinese (Windows only)
2   euc-cn      simplified Chinese (Unix only)
2   cp950       traditional Chinese (on Unix alias for big5)
2   big5        traditional Chinese (on Windows alias for cp950)
2   euc-tw      traditional Chinese (Unix only)
2   2byte-{name} Unix: any double-byte encoding (Vim specific name)
2   cp{number}  MS-Windows: any installed double-byte codepage
u   utf-8       32 bit UTF-8 encoded Unicode (ISO/IEC 10646-1)
u   ucs-2       16 bit UCS-2 encoded Unicode (ISO/IEC 10646-1)
u   ucs-2le     like ucs-2, little endian
u   utf-16      ucs-2 extended with double-words for more characters
u   utf-16le    like utf-16, little endian
u   ucs-4       32 bit UCS-4 encoded Unicode (ISO/IEC 10646-1)
u   ucs-4le     like ucs-4, little endian

The {name} can be any encoding name that your system supports.  It is passed
to iconv() to convert between the encoding of the file and the current locale.
For MS-Windows "cp{number}" means using codepage {number}.
Examples: >
                :set encoding=8bit-cp1252
                :set encoding=2byte-cp932

The MS-Windows codepage 1252 is very similar to latin1.  For practical reasons
the same encoding is used and it's called latin1.  'isprint' can be used to
display the characters 0x80 - 0xA0 or not.

Several aliases can be used, they are translated to one of the names above.
An incomplete list:

1   ansi        same as latin1 (obsolete, for backward compatibility)
2   japan       Japanese: on Unix "euc-jp", on MS-Windows cp932
2   korea       Korean: on Unix "euc-kr", on MS-Windows cp949
2   prc         simplified Chinese: on Unix "euc-cn", on MS-Windows cp936
2   chinese     same as "prc"
2   taiwan      traditional Chinese: on Unix "euc-tw", on MS-Windows cp950
u   utf8        same as utf-8
u   unicode     same as ucs-2
u   ucs2be      same as ucs-2 (big endian)
u   ucs-2be     same as ucs-2 (big endian)
u   ucs-4be     same as ucs-4 (big endian)
u   utf-32      same as ucs-4
u   utf-32le    same as ucs-4le
    default     stands for the default value of 'encoding', depends on the
                environment

For the UCS codes the byte order matters.  This is tricky, use UTF-8 whenever
you can.  The default is to use big-endian (most significant byte comes
first):
```

name	bytes	char ~
ucs-2	11 22	1122
ucs-2le	22 11	1122
ucs-4	11 22 33 44	11223344
ucs-4le	44 33 22 11	11223344

On MS-Windows systems you often want to use "ucs-2le", because it uses little
endian UCS-2.

There are a few encodings which are similar, but not exactly the same. Vim
treats them as if they were different encodings, so that conversion will be
done when needed. You might want to use the similar name to avoid conversion
or when conversion is not possible:

```
        cp932, shift-jis, sjis
        cp936, euc-cn
```

encoding-table

Normally 'encoding' is equal to your current locale and 'termencoding' is
empty. This means that your keyboard and display work with characters encoded
in your current locale, and Vim uses the same characters internally.

You can make Vim use characters in a different encoding by setting the
'encoding' option to a different value. Since the keyboard and display still
use the current locale, conversion needs to be done. The 'termencoding' then
takes over the value of the current locale, so Vim converts between 'encoding'
and 'termencoding'. Example: >

```
        :let &termencoding = &encoding
        :set encoding=utf-8
```

However, not all combinations of values are possible. The table below tells
you how each of the nine combinations works. This is further restricted by
not all conversions being possible, iconv() being present, etc. Since this
depends on the system used, no detailed list can be given.

('tenc' is the short name for 'termencoding' and 'enc' short for 'encoding')

'tenc'	'enc'	remark ~
8bit	8bit	Works. When 'termencoding' is different from 'encoding' typing and displaying may be wrong for some characters, Vim does NOT perform conversion (set 'encoding' to "utf-8" to get this).
8bit	2byte	MS-Windows: works for all codepages installed on your system; you can only type 8bit characters; Other systems: does NOT work.
8bit	Unicode	Works, but only 8bit characters can be typed directly (others through digraphs, keymaps, etc.); in a terminal you can only see 8bit characters; the GUI can show all characters that the 'guifont' supports.
2byte	8bit	Works, but typing non-ASCII characters might be a problem.
2byte	2byte	MS-Windows: works for all codepages installed on your system; typing characters might be a problem when locale is different from 'encoding'. Other systems: Only works when 'termencoding' is equal to 'encoding', you might as well leave it empty.
2byte	Unicode	works, Vim will translate typed characters.
Unicode	8bit	works (unusual)
Unicode	2byte	does NOT work

```
 Unicode   Unicode      works very well (leaving 'termencoding' empty works
                        the same way, because all Unicode is handled
                        internally as UTF-8)

CONVERSION                                                *charset-conversion*

Vim will automatically convert from one to another encoding in several places:
- When reading a file and 'fileencoding' is different from 'encoding'
- When writing a file and 'fileencoding' is different from 'encoding'
- When displaying characters and 'termencoding' is different from 'encoding'
- When reading input and 'termencoding' is different from 'encoding'
- When displaying messages and the encoding used for LC_MESSAGES differs from
  'encoding' (requires a gettext version that supports this).
- When reading a Vim script where |:scriptencoding| is different from
  'encoding'.
- When reading or writing a |viminfo| file.
Most of these require the |+iconv| feature.  Conversion for reading and
writing files may also be specified with the 'charconvert' option.

Useful utilities for converting the charset:
    All:            iconv
        GNU iconv can convert most encodings.  Unicode is used as the
        intermediate encoding, which allows conversion from and to all other
        encodings.  See http://www.gnu.org/directory/libiconv.html.

    Japanese:       nkf
        Nkf is "Network Kanji code conversion Filter".  One of the most unique
        facility of nkf is the guess of the input Kanji code.  So, you don't
        need to know what the inputting file's |charset| is.  When convert to
        EUC-JP from ISO-2022-JP or Shift_JIS, simply do the following command
        in Vim:
            :%!nkf -e
        Nkf can be found at:
        http://www.sfc.wide.ad.jp/~max/FreeBSD/ports/distfiles/nkf-1.62.tar.gz

    Chinese:        hc
        Hc is "Hanzi Converter".  Hc convert a GB file to a Big5 file, or Big5
        file to GB file.  Hc can be found at:
        ftp://ftp.cuhk.hk/pub/chinese/ifcss/software/unix/convert/hc-30.tar.gz

    Korean:         hmconv
        Hmconv is Korean code conversion utility especially for E-mail.  It can
        convert between EUC-KR and ISO-2022-KR.  Hmconv can be found at:
        ftp://ftp.kaist.ac.kr/pub/hangul/code/hmconv/

    Multilingual:   lv
        Lv is a Powerful Multilingual File Viewer.  And it can be worked as
        |charset| converter.  Supported |charset|: ISO-2022-CN, ISO-2022-JP,
        ISO-2022-KR, EUC-CN, EUC-JP, EUC-KR, EUC-TW, UTF-7, UTF-8, ISO-8859
        series, Shift_JIS, Big5 and HZ.  Lv can be found at:
        http://www.ff.iij4u.or.jp/~nrt/lv/index.html


                                                        *mbyte-conversion*
When reading and writing files in an encoding different from 'encoding',
conversion needs to be done.  These conversions are supported:
- All conversions between Latin-1 (ISO-8859-1), UTF-8, UCS-2 and UCS-4 are
  handled internally.
- For MS-Windows, when 'encoding' is a Unicode encoding, conversion from and
  to any codepage should work.
- Conversion specified with 'charconvert'
- Conversion with the iconv library, if it is available.
```

```
        Old versions of GNU iconv() may cause the conversion to fail (they
        request a very large buffer, more than Vim is willing to provide).
        Try getting another iconv() implementation.

                                                        *iconv-dynamic*
On MS-Windows Vim can be compiled with the |+iconv/dyn| feature.  This means
Vim will search for the "iconv.dll" and "libiconv.dll" libraries.  When
neither of them can be found Vim will still work but some conversions won't be
possible.

==============================================================================
4. Using a terminal                                     *mbyte-terminal*

The GUI fully supports multi-byte characters.  It is also possible in a
terminal, if the terminal supports the same encoding that Vim uses.  Thus this
is less flexible.

For example, you can run Vim in a xterm with added multi-byte support and/or
|XIM|.  Examples are kterm (Kanji term) and hanterm (for Korean), Eterm
(Enlightened terminal) and rxvt.

If your terminal does not support the right encoding, you can set the
'termencoding' option.  Vim will then convert the typed characters from
'termencoding' to 'encoding'.  And displayed text will be converted from
'encoding' to 'termencoding'.  If the encoding supported by the terminal
doesn't include all the characters that Vim uses, this leads to lost
characters.  This may mess up the display.  If you use a terminal that
supports Unicode, such as the xterm mentioned below, it should work just fine,
since nearly every character set can be converted to Unicode without loss of
information.


UTF-8 IN XFREE86 XTERM                                  *UTF8-xterm*

This is a short explanation of how to use UTF-8 character encoding in the
xterm that comes with XFree86 by Thomas Dickey (text by Markus Kuhn).

Get the latest xterm version which has now UTF-8 support:

        http://invisible-island.net/xterm/xterm.html

Compile it with "./configure --enable-wide-chars ; make"

Also get the ISO 10646-1 version of various fonts, which is available on

        http://www.cl.cam.ac.uk/~mgk25/download/ucs-fonts.tar.gz

and install the font as described in the README file.

Now start xterm with >

  xterm -u8 -fn -misc-fixed-medium-r-semicondensed--13-120-75-75-c-60-iso10646-1
or, for bigger character: >
  xterm -u8 -fn -misc-fixed-medium-r-normal--15-140-75-75-c-90-iso10646-1

and you will have a working UTF-8 terminal emulator.  Try both >

   cat utf-8-demo.txt
   vim utf-8-demo.txt

with the demo text that comes with ucs-fonts.tar.gz in order to see
whether there are any problems with UTF-8 in your xterm.
```

```
For Vim you may need to set 'encoding' to "utf-8".

==============================================================================
5.  Fonts on X11                                        *mbyte-fonts-X11*

Unfortunately, using fonts in X11 is complicated.  The name of a single-byte
font is a long string.  For multi-byte fonts we need several of these...

Note: Most of this is no longer relevant for GTK+ 2.  Selecting a font via
its XLFD is not supported; see 'guifont' for an example of how to
set the font.  Do yourself a favor and ignore the |XLFD| and |xfontset|
sections below.

First of all, Vim only accepts fixed-width fonts for displaying text.  You
cannot use proportionally spaced fonts.  This excludes many of the available
(and nicer looking) fonts.  However, for menus and tooltips any font can be
used.

Note that Display and Input are independent.  It is possible to see your
language even though you have no input method for it.

You should get a default font for menus and tooltips that works, but it might
be ugly.  Read the following to find out how to select a better font.


X LOGICAL FONT DESCRIPTION (XLFD)
                                                        *XLFD*
XLFD is the X font name and contains the information about the font size,
charset, etc.  The name is in this format:

FOUNDRY-FAMILY-WEIGHT-SLANT-WIDTH-STYLE-PIXEL-POINT-X-Y-SPACE-AVE-CR-CE

Each field means:

- FOUNDRY:  FOUNDRY field.  The company that created the font.
- FAMILY:   FAMILY_NAME field.  Basic font family name.  (helvetica, gothic,
            times, etc)
- WEIGHT:   WEIGHT_NAME field.  How thick the letters are.  (light, medium,
            bold, etc)
- SLANT:    SLANT field.
                r:  Roman (no slant)
                i:  Italic
                o:  Oblique
                ri: Reverse Italic
                ro: Reverse Oblique
                ot: Other
                number: Scaled font
- WIDTH:    SETWIDTH_NAME field.  Width of characters.  (normal, condensed,
            narrow, double wide)
- STYLE:    ADD_STYLE_NAME field.  Extra info to describe font.  (Serif, Sans
            Serif, Informal, Decorated, etc)
- PIXEL:    PIXEL_SIZE field.  Height, in pixels, of characters.
- POINT:    POINT_SIZE field.  Ten times height of characters in points.
- X:        RESOLUTION_X field.  X resolution (dots per inch).
- Y:        RESOLUTION_Y field.  Y resolution (dots per inch).
- SPACE:    SPACING field.
                p:  Proportional
                m:  Monospaced
                c:  CharCell
- AVE:      AVERAGE_WIDTH field.  Ten times average width in pixels.
- CR:       CHARSET_REGISTRY field.  The name of the charset group.
- CE:       CHARSET_ENCODING field.  The rest of the charset name.  For some
            charsets, such as JIS X 0208, if this field is 0, code points has
            the same value as GL, and GR if 1.
```

For example, in case of a 16 dots font corresponding to JIS X 0208, it is written like:

```
    -misc-fixed-medium-r-normal--16-110-100-100-c-160-jisx0208.1990-0
```

X FONTSET

fontset *xfontset*

A single-byte charset is typically associated with one font. For multi-byte charsets a combination of fonts is often used. This means that one group of characters are used from one font and another group from another font (which might be double wide). This collection of fonts is called a fontset.

Which fonts are required in a fontset depends on the current locale. X windows maintains a table of which groups of characters are required for a locale. You have to specify all the fonts that a locale requires in the 'guifontset' option.

NOTE: The fontset always uses the current locale, even though 'encoding' may be set to use a different charset. In that situation you might want to use 'guifont' and 'guifontwide' instead of 'guifontset'.

Example:

|charset|	language	"groups of characters" ~
GB2312	Chinese (simplified)	ISO-8859-1 and GB 2312
Big5	Chinese (traditional)	ISO-8859-1 and Big5
CNS-11643	Chinese (traditional)	ISO-8859-1, CNS 11643-1 and CNS 11643-2
EUC-JP	Japanese	JIS X 0201 and JIS X 0208
EUC-KR	Korean	ISO-8859-1 and KS C 5601 (KS X 1001)

You can search for fonts using the xlsfonts command. For example, when you're searching for a font for KS C 5601: >

```
    xlsfonts | grep ksc5601
```

This is complicated and confusing. You might want to consult the X-Windows documentation if there is something you don't understand.

base_font_name_list

When you have found the names of the fonts you want to use, you need to set the 'guifontset' option. You specify the list by concatenating the font names and putting a comma in between them.

For example, when you use the ja_JP.eucJP locale, this requires JIS X 0201 and JIS X 0208. You could supply a list of fonts that explicitly specifies the charsets, like: >

```
 :set guifontset=-misc-fixed-medium-r-normal--14-130-75-75-c-140-jisx0208.1983-0,
        \-misc-fixed-medium-r-normal--14-130-75-75-c-70-jisx0201.1976-0
```

Alternatively, you can supply a base font name list that omits the charset name, letting X-Windows select font characters required for the locale. For example: >

```
 :set guifontset=-misc-fixed-medium-r-normal--14-130-75-75-c-140,
        \-misc-fixed-medium-r-normal--14-130-75-75-c-70
```

Alternatively, you can supply a single base font name that allows X-Windows to select from all available fonts. For example: >

```
 :set guifontset=-misc-fixed-medium-r-normal--14-*

Alternatively, you can specify alias names.  See the fonts.alias file in the
fonts directory (e.g., /usr/X11R6/lib/X11/fonts/).  For example: >

 :set guifontset=k14,r14
<
                                                        *E253*
Note that in East Asian fonts, the standard character cell is square.  When
mixing a Latin font and an East Asian font, the East Asian font width should
be twice the Latin font width.

If 'guifontset' is not empty, the "font" argument of the |:highlight| command
is also interpreted as a fontset.  For example, you should use for
highlighting: >
        :hi Comment font=english_font,your_font
If you use a wrong "font" argument you will get an error message.
Also make sure that you set 'guifontset' before setting fonts for highlight
groups.


USING RESOURCE FILES

Instead of specifying 'guifontset', you can set X11 resources and Vim will
pick them up.  This is only for people who know how X resource files work.

For Motif and Athena insert these three lines in your $HOME/.Xdefaults file:

        Vim.font: |base_font_name_list|
        Vim*fontSet: |base_font_name_list|
        Vim*fontList: your_language_font

Note: Vim.font is for text area.
      Vim*fontSet is for menu.
      Vim*fontList is for menu (for Motif GUI)

For example, when you are using Japanese and a 14 dots font, >

        Vim.font: -misc-fixed-medium-r-normal--14-*
        Vim*fontSet: -misc-fixed-medium-r-normal--14-*
        Vim*fontList: -misc-fixed-medium-r-normal--14-*
<
or: >

        Vim*font: k14,r14
        Vim*fontSet: k14,r14
        Vim*fontList: k14,r14
<
To have them take effect immediately you will have to do >

        xrdb -merge ~/.Xdefaults

Otherwise you will have to stop and restart the X server before the changes
take effect.


The GTK+ version of GUI Vim does not use .Xdefaults, use ~/.gtkrc instead.
The default mostly works OK.  But for the menus you might have to change
it.  Example: >

        style "default"
        {
                fontset="-*-*-medium-r-normal--14-*-*-*-c-*-*-*"
        }
        widget_class "*" style "default"
```

==

6. Fonts on MS-Windows   *mbyte-fonts-MSwin*

The simplest is to use the font dialog to select fonts and try them out. You can find this at the "Edit/Select Font..." menu. Once you find a font name that works well you can use this command to see its name: >

```
        :set guifont
```

Then add a command to your |gvimrc| file to set 'guifont': >

```
        :set guifont=courier_new:h12
```

==

7. Input on X11   *mbyte-XIM*

X INPUT METHOD (XIM) BACKGROUND   *XIM* *xim* *x-input-method*

XIM is an international input module for X. There are two kinds of structures, Xlib unit type and |IM-server| (Input-Method server) type. |IM-server| type is suitable for complex input, such as CJK.

- IM-server
                                                        IM-server
  In |IM-server| type input structures, the input event is handled by either of the two ways: FrontEnd system and BackEnd system. In the FrontEnd system, input events are snatched by the |IM-server| first, then |IM-server| give the application the result of input. On the other hand, the BackEnd system works reverse order. MS Windows adopt BackEnd system. In X, most of |IM-server|s adopt FrontEnd system. The demerit of BackEnd system is the large overhead in communication, but it provides safe synchronization with no restrictions on applications.

  For example, there are xwnmo and kinput2 Japanese |IM-server|, both are FrontEnd system. Xwnmo is distributed with Wnn (see below), kinput2 can be found at: ftp://ftp.sra.co.jp/pub/x11/kinput2/

  For Chinese, there's a great XIM server named "xcin", you can input both Traditional and Simplified Chinese characters. And it can accept other locale if you make a correct input table. Xcin can be found at:
  http://cle.linux.org.tw/xcin/
  Others are scim: http://scim.freedesktop.org/ and fcitx:
  http://www.fcitx.org/

- Conversion Server
                                                        conversion-server
  Some system needs additional server: conversion server. Most of Japanese |IM-server|s need it, Kana-Kanji conversion server. For Chinese inputting, it depends on the method of inputting, in some methods, PinYin or ZhuYin to HanZi conversion server is needed. For Korean inputting, if you want to input Hanja, Hangul-Hanja conversion server is needed.

  For example, the Japanese inputting process is divided into 2 steps. First we pre-input Hira-gana, second Kana-Kanji conversion. There are so many Kanji characters (6349 Kanji characters are defined in JIS X 0208) and the number of Hira-gana characters are 76. So, first, we pre-input text as pronounced in Hira-gana, second, we convert Hira-gana to Kanji or Kata-Kana, if needed. There are some Kana-Kanji conversion server: jserver

```
  (distributed with Wnn, see below) and canna.  Canna can be found at:
  http://canna.sourceforge.jp/

There is a good input system: Wnn4.2.  Wnn 4.2 contains,
    xwnmo (|IM-server|)
    jserver (Japanese Kana-Kanji conversion server)
    cserver (Chinese PinYin or ZhuYin to simplified HanZi conversion server)
    tserver (Chinese PinYin or ZhuYin to traditional HanZi conversion server)
    kserver (Hangul-Hanja conversion server)
Wnn 4.2 for several systems can be found at various places on the internet.
Use the RPM or port for your system.


- Input Style
                                                        *xim-input-style*
  When inputting CJK, there are four areas:
      1. The area to display of the input while it is being composed
      2. The area to display the currently active input mode.
      3. The area to display the next candidate for the selection.
      4. The area to display other tools.

  The third area is needed when converting.  For example, in Japanese
  inputting, multiple Kanji characters could have the same pronunciation, so
  a sequence of Hira-gana characters could map to a distinct sequence of Kanji
  characters.

  The first and second areas are defined in international input of X with the
  names of "Preedit Area", "Status Area" respectively.  The third and fourth
  areas are not defined and are left to be managed by the |IM-server|.  In the
  international input, four input styles have been defined using combinations
  of Preedit Area and Status Area: |OnTheSpot|, |OffTheSpot|, |OverTheSpot|
  and |Root|.

  Currently, GUI Vim supports three styles, |OverTheSpot|, |OffTheSpot| and
  |Root|.
  When compiled with |+GUI_GTK| feature, GUI Vim supports two styles,
  |OnTheSpot| and |OverTheSpot|.  You can select the style with the 'imstyle'
  option.

*.  on-the-spot                                         *OnTheSpot*
    Preedit Area and Status Area are performed by the client application in
    the area of application.  The client application is directed by the
    |IM-server| to display all pre-edit data at the location of text
    insertion.  The client registers callbacks invoked by the input method
    during pre-editing.
*.  over-the-spot                                       *OverTheSpot*
    Status Area is created in a fixed position within the area of application,
    in case of Vim, the position is the additional status line.  Preedit Area
    is made at present input position of application.  The input method
    displays pre-edit data in a window which it brings up directly over the
    text insertion position.
*.  off-the-spot                                        *OffTheSpot*
    Preedit Area and Status Area are performed in the area of application, in
    case of Vim, the area is additional status line.  The client application
    provides display windows for the pre-edit data to the input method which
    displays into them directly.
*.  root-window                                         *Root*
    Preedit Area and Status Area are outside of the application.  The input
    method displays all pre-edit data in a separate area of the screen in a
    window specific to the input method.
```

```
USING XIM                       *multibyte-input* *E284* *E286* *E287* *E288*
                                *E285* *E289*

Note that Display and Input are independent.  It is possible to see your
language even though you have no input method for it.  But when your Display
method doesn't match your Input method, the text will be displayed wrong.

        Note: You can not use IM unless you specify 'guifontset'.
              Therefore, Latin users, you have to also use 'guifontset'
              if you use IM.

To input your language you should run the |IM-server| which supports your
language and |conversion-server| if needed.

The next 3 lines should be put in your ~/.Xdefaults file.  They are common for
all X applications which uses |XIM|.  If you already use |XIM|, you can skip
this. >

        *international: True
        *.inputMethod: your_input_server_name
        *.preeditType: your_input_style
<
input_server_name       is your |IM-server| name (check your |IM-server|
                        manual).
your_input_style        is one of |OverTheSpot|, |OffTheSpot|, |Root|.  See
                        also |xim-input-style|.

*international may not necessary if you use X11R6.
*.inputMethod and *.preeditType are optional if you use X11R6.

For example, when you are using kinput2 as |IM-server|, >

        *international: True
        *.inputMethod: kinput2
        *.preeditType: OverTheSpot
<
When using |OverTheSpot|, GUI Vim always connects to the IM Server even in
Normal mode, so you can input your language with commands like "f" and "r".
But when using one of the other two methods, GUI Vim connects to the IM Server
only if it is not in Normal mode.

If your IM Server does not support |OverTheSpot|, and if you want to use your
language with some Normal mode command like "f" or "r", then you should use a
localized xterm  or an xterm which supports |XIM|

If needed, you can set the XMODIFIERS environment variable:

        sh:  export XMODIFIERS="@im=input_server_name"
        csh: setenv XMODIFIERS "@im=input_server_name"

For example, when you are using kinput2 as |IM-server| and sh, >

        export XMODIFIERS="@im=kinput2"
<

FULLY CONTROLLED XIM

You can fully control XIM, like with IME of MS-Windows (see |multibyte-ime|).
This is currently only available for the GTK GUI.

Before using fully controlled XIM, one setting is required.  Set the
'imactivatekey' option to the key that is used for the activation of the input
```

```
method.  For example, when you are using kinput2 + canna as IM Server, the
activation key is probably Shift+Space: >

        :set imactivatekey=S-space

See 'imactivatekey' for the format.

==============================================================================
8.  Input on MS-Windows                                  *mbyte-IME*

(Windows IME support)                            *multibyte-ime* *IME*

{only works Windows GUI and compiled with the |+multi_byte_ime| feature}

To input multibyte characters on Windows, you can use an Input Method Editor
(IME).  In process of your editing text, you must switch status (on/off) of
IME many many many times.  Because IME with status on is hooking all of your
key inputs, you cannot input 'j', 'k', or almost all of keys to Vim directly.

This |+multi_byte_ime| feature help this.  It reduce times of switch status of
IME manually.  In normal mode, there are almost no need working IME, even
editing multibyte text.  So exiting insert mode with ESC, Vim memorize last
status of IME and force turn off IME.  When re-enter insert mode, Vim revert
IME status to that memorized automatically.

This works on not only insert-normal mode, but also search-command input and
replace mode.
The options 'iminsert', 'imsearch' and 'imcmdline' can be used to chose
the different input methods or disable them temporarily.

WHAT IS IME
    IME is a part of East asian version Windows.  That helps you to input
    multibyte character.  English and other language version Windows does not
    have any IME.  (Also there is no need usually.) But there is one that
    called Microsoft Global IME.  Global IME is a part of Internet Explorer
    4.0 or above.  You can get more information about Global IME, at below
    URL.

WHAT IS GLOBAL IME                                       *global-ime*
    Global IME makes capability to input Chinese, Japanese, and Korean text
    into Vim buffer on any language version of Windows 98, Windows 95, and
    Windows NT 4.0.
    On Windows 2000 and XP it should work as well (without downloading).  On
    Windows 2000 Professional, Global IME is built in, and the Input Locales
    can be added through Control Panel/Regional Options/Input Locales.
    Please see below URL for detail of Global IME.  You can also find various
    language version of Global IME at same place.

    - Global IME detailed information.
        http://search.microsoft.com/results.aspx?q=global+ime

    - Active Input Method Manager (Global IME)
        http://msdn.microsoft.com/en-us/library/aa741221(v=VS.85).aspx

    Support for Global IME is an experimental feature.

NOTE: For IME to work you must make sure the input locales of your language
are added to your system.  The exact location of this depends on the version
of Windows you use.  For example, on my Windows 2000 box:
1. Control Panel
2. Regional Options
3. Input Locales Tab
```

```
4. Add Installed input locales -> Chinese(PRC)
   The default is still English (United Stated)


Cursor color when IME or XIM is on                              *CursorIM*
    There is a little cute feature for IME.  Cursor can indicate status of IME
    by changing its color.  Usually status of IME was indicated by little icon
    at a corner of desktop (or taskbar).  It is not easy to verify status of
    IME.  But this feature help this.
    This works in the same way when using XIM.

    You can select cursor color when status is on by using highlight group
    CursorIM.  For example, add these lines to your |gvimrc|: >

        if has('multi_byte_ime')
            highlight Cursor guifg=NONE guibg=Green
            highlight CursorIM guifg=NONE guibg=Purple
        endif
<
    Cursor color with off IME is green.  And purple cursor indicates that
    status is on.

==============================================================================
9. Input with a keymap                                  *mbyte-keymap*

When the keyboard doesn't produce the characters you want to enter in your
text, you can use the 'keymap' option.  This will translate one or more
(English) characters to another (non-English) character.  This only happens
when typing text, not when typing Vim commands.  This avoids having to switch
between two keyboard settings.
{only available when compiled with the |+keymap| feature}

The value of the 'keymap' option specifies a keymap file to use.  The name of
this file is one of these two:

        keymap/{keymap}_{encoding}.vim
        keymap/{keymap}.vim

Here {keymap} is the value of the 'keymap' option and {encoding} of the
'encoding' option.  The file name with the {encoding} included is tried first.

'runtimepath' is used to find these files.  To see an overview of all
available keymap files, use this: >
        :echo globpath(&rtp, "keymap/*.vim")

In Insert and Command-line mode you can use CTRL-^ to toggle between using the
keyboard map or not. |i_CTRL-^| |c_CTRL-^|
This flag is remembered for Insert mode with the 'iminsert' option.  When
leaving and entering Insert mode the previous value is used.  The same value
is also used for commands that take a single character argument, like |f| and
|r|.
For Command-line mode the flag is NOT remembered.  You are expected to type an
Ex command first, which is ASCII.
For typing search patterns the 'imsearch' option is used.  It can be set to
use the same value as for 'iminsert'.
                                                                *lCursor*
It is possible to give the GUI cursor another color when the language mappings
are being used.  This is disabled by default, to avoid that the cursor becomes
invisible when you use a non-standard background color.  Here is an example to
use a brightly colored cursor: >
        :highlight Cursor guifg=NONE guibg=Green
        :highlight lCursor guifg=NONE guibg=Cyan
```

<
                keymap-file-format *:loadk* *:loadkeymap* *E105* *E791*

The keymap file looks something like this: >

```
        " Maintainer:   name <email@address>
        " Last Changed: 2001 Jan 1

        let b:keymap_name = "short"

        loadkeymap
        a       A
        b       B       comment
```

The lines starting with a " are comments and will be ignored. Blank lines are also ignored. The lines with the mappings may have a comment after the useful text.

The "b:keymap_name" can be set to a short name, which will be shown in the status line. The idea is that this takes less room than the value of 'keymap', which might be long to distinguish between different languages, keyboards and encodings.

The actual mappings are in the lines below "loadkeymap". In the example "a" is mapped to "A" and "b" to "B". Thus the first item is mapped to the second item. This is done for each line, until the end of the file.
These items are exactly the same as what can be used in a |:lnoremap| command, using "<buffer>" to make the mappings local to the buffer.
You can check the result with this command: >

```
        :lmap
```

The two items must be separated by white space. You cannot include white space inside an item, use the special names "<Tab>" and "<Space>" instead.
The length of the two items together must not exceed 200 bytes.

It's possible to have more than one character in the first column. This works like a dead key. Example: >

```
        'a      á
```

Since Vim doesn't know if the next character after a quote is really an "a", it will wait for the next character. To be able to insert a single quote, also add this line: >

```
        ''      '
```

Since the mapping is defined with |:lnoremap| the resulting quote will not be used for the start of another character.
The "accents" keymap uses this.   *keymap-accents*

The first column can also be in |<>| form:

```
        <C-c>           Ctrl-C
        <A-c>           Alt-c
        <A-C>           Alt-C
```

Note that the Alt mappings may not work, depending on your keyboard and terminal.

Although it's possible to have more than one character in the second column, this is unusual. But you can use various ways to specify the character: >

```
        A       a               literal character
        A       <char-97>       decimal value
        A       <char-0x61>     hexadecimal value
        A       <char-0141>     octal value
        x       <Space>         special key name
```

The characters are assumed to be encoded for the current value of 'encoding'.
It's possible to use ":scriptencoding" when all characters are given literally. That doesn't work when using the <char-> construct, because the

```
conversion is done on the keymap file, not on the resulting character.

The lines after "loadkeymap" are interpreted with 'cpoptions' set to "C".
This means that continuation lines are not used and a backslash has a special
meaning in the mappings.  Examples: >

        " a comment line
        \"      x       maps " to x
        \\      y       maps \ to y

If you write a keymap file that will be useful for others, consider submitting
it to the Vim maintainer for inclusion in the distribution:
<maintainer@vim.org>


HEBREW KEYMAP                                           *keymap-hebrew*

This file explains what characters are available in UTF-8 and CP1255 encodings,
and what the keymaps are to get those characters:

glyph   encoding           keymap ~
Char   utf-8 cp1255  hebrew  hebrewp  name ~
                                                    a     'alef  0x5d0  0xe0     t    א
                                                      b     bet  0x5d1  0xe1     c    ב
                                                    g     gimel  0x5d2  0xe2     d    ג
                                                    d     dalet  0x5d3  0xe3     s    ד
                                                       h     he  0x5d4  0xe4     v    ה
                                                      v     vav  0x5d5  0xe5     u    ו
                                                    z     zayin  0x5d6  0xe6     z    ז
                                                      j     het  0x5d7  0xe7     j    ח
                                                      T     tet  0x5d8  0xe8     y    ט
                                                      y     yod  0x5d9  0xe9     h    י
                                                K     kaf sofit  0x5da  0xea     l    ך
                                                      k     kaf  0x5db  0xeb     f    כ
                                                    l     lamed  0x5dc  0xec     k    ל
                                                M     mem sofit  0x5dd  0xed     o    ם
                                                      m     mem  0x5de  0xee     n    מ
                                                N     nun sofit  0x5df  0xef     i    ן
                                                      n     nun  0x5e0  0xf0     b    נ
                                                   s     samech  0x5e1  0xf1     x    ס
                                                    u     `ayin  0x5e2  0xf2     g    ע
                                                 P     pe sofit  0x5e3  0xf3     ;    ף
                                                       p     pe  0x5e4  0xf4     p    פ
                                              X     tsadi sofit  0x5e5  0xf5     .    ץ
                                                    x     tsadi  0x5e6  0xf6     m    צ
                                                      q     qof  0x5e7  0xf7     e    ק
                                                     r     resh  0x5e8  0xf8     r    ר
                                                     w     shin  0x5e9  0xf9     a    ש
                                                      t     tav  0x5ea  0xfa     ,    ת

Vowel marks and special punctuation:
                                                0x5b0  0xc0     A:      A:   sheva    הְ
                                          0x5b1  0xc1     HE      HE   hataf segol    הֱ
                                          0x5b2  0xc2     HA      HA   hataf patah    הֲ
                                         0x5b3  0xc3     HO      HO   hataf qamats    הֳ
                                                0x5b4  0xc4     I       I    hiriq    הִ
                                                0x5b5  0xc5     AY      AY   tsere    הֵ
                                                0x5b6  0xc6     E       E    segol    הֶ
                                                0x5b7  0xc7     AA      AA   patah    הַ
                                               0x5b8  0xc8     AO      AO   qamats    הָ
                                                0x5b9  0xc9     O       O    holam    הֹ
                                               0x5bb  0xcb     U       U    qubuts    הֻ
```

```
                                                  0x5bc  0xcc     D       D    dagesh    בּ
                                                   0x5bd  0xcd     ]T      ]T   meteg    הֽ
                                                   0x5be  0xce     ]Q      ]Q   maqaf   ־ה
                                                    0x5bf  0xcf     ]R      ]R   rafe    בֿ
                                                   0x5c0  0xd0     ]p      ]p   paseq   ׀ב
                                                0x5c1  0xd1     SR      SR   shin-dot    שׁ
                                                 0x5c2  0xd2     SL      SL   sin-dot    שׂ
                                               0x5c3  0xd3     ]P      ]P   sof-pasuq    ׃
                                              0x5f0  0xd4     VV      VV   double-vav    װ
                                                 0x5f1  0xd5     VY      VY   vav-yod    ױ
                                                 0x5f2  0xd6     YY      YY   yod-yod    ײ

The following are only available in utf-8

Cantillation marks:
glyph
Char utf-8 hebrew name
                                                                 0x591   C:   etnahta  ב֑
                                                                   0x592   Cs   segol  ב֒
                                                              0x593   CS   shalshelet  ב֓
                                                             0x594   Cz   zaqef qatan  ב֔
                                                             0x595   CZ   zaqef gadol  ב֕
                                                                  0x596   Ct   tipeha  ב֖
                                                                   0x597   Cr   revia  ב֗
                                                                   0x598   Cq   zarqa  ב֘
                                                                  0x599   Cp   pashta  ב֙
                                                                   0x59a   C!   yetiv  ב֚
                                                                   0x59b   Cv   tevir  ב֛
                                                                  0x59c   Cg   geresh  ב֜
                                                            0x59d   C*   geresh qadim  ב֝
                                                               0x59e   CG   gershayim  ב֞
                                                            0x59f   CP   qarnei-parah  ב֟
                                                         0x5aa   Cy   yerach-ben-yomo  ב֪
                                                                     0x5ab   Co   ole  ב֫
                                                                    0x5ac   Ci   iluy  ב֬
                                                                    0x5ad   Cd   dehi  ב֭
                                                                   0x5ae   Cn   zinor  ב֮
                                                           0x5af   CC   masora circle  ב֯

Combining forms:
                                                0xfb20  X`   Alternative `ayin    ﬠ
                                                0xfb21  X'   Alternative 'alef    ﬡ
                                                0xfb22  X-d  Alternative dalet    ﬢ
                                                   0xfb23  X-h  Alternative he    ﬣ
                                                  0xfb24  X-k  Alternative kaf    ﬤ
                                                0xfb25  X-l  Alternative lamed    ﬥ
                                            0xfb26  X-m  Alternative mem-sofit    ﬦ
                                                 0xfb27  X-r  Alternative resh    ﬧ
                                                  0xfb28  X-t  Alternative tav    ﬨ
﬩    0xfb29  X-+  Alternative plus
                                                    0xfb2a  XW   shin+shin-dot    שׁ
                                                     0xfb2b  Xw   shin+sin-dot    שׂ
                                            0xfb2c  X..W  shin+shin-dot+dagesh    שּׁ
                                             0xfb2d  X..w  shin+sin-dot+dagesh    שּׂ
                                                        0xfb2e  XA   alef+patah    אַ
                                                       0xfb2f  XO   alef+qamats    אָ
                                                 (0xfb30  XI   alef+hiriq (mapiq    אּ
                                                        0xfb31  X.b  bet+dagesh    בּ
                                                      0xfb32  X.g  gimel+dagesh    גּ
                                                      0xfb33  X.d  dalet+dagesh    דּ
                                                         0xfb34  X.h  he+dagesh    הּ
                                                         0xfb35  Xu  vav+dagesh    וּ
```

0xfb36	X.z	zayin+dagesh	זּ
0xfb38	X.T	tet+dagesh	טּ
0xfb39	X.y	yud+dagesh	יּ
0xfb3a	X.K	kaf sofit+dagesh	ךּ
0xfb3b	X.k	kaf+dagesh	כּ
0xfb3c	X.l	lamed+dagesh	לּ
0xfb3e	X.m	mem+dagesh	מּ
0xfb40	X.n	nun+dagesh	נּ
0xfb41	X.s	samech+dagesh	סּ
0xfb43	X.P	pe sofit+dagesh	ףּ
0xfb44	X.p	pe+dagesh	פּ
0xfb46	X.x	tsadi+dagesh	צּ
0xfb47	X.q	qof+dagesh	קּ
0xfb48	X.r	resh+dagesh	רּ
0xfb49	X.w	shin+dagesh	שּ
0xfb4a	X.t	tav+dagesh	תּ
0xfb4b	Xo	vav+holam	וֹ
0xfb4c	XRb	bet+rafe	בֿ
0xfb4d	XRk	kaf+rafe	כֿ
0xfb4e	XRp	pe+rafe	פֿ
0xfb4f	Xal	alef-lamed	ﭏ

```
==============================================================================
10. Using UTF-8                         *mbyte-utf8* *UTF-8* *utf-8* *utf8*
                                                    *Unicode* *unicode*
```

The Unicode character set was designed to include all characters from other character sets. Therefore it is possible to write text in any language using Unicode (with a few rarely used languages excluded). And it's mostly possible to mix these languages in one file, which is impossible with other encodings.

Unicode can be encoded in several ways. The most popular one is UTF-8, which uses one or more bytes for each character and is backwards compatible with ASCII. On MS-Windows UTF-16 is also used (previously UCS-2), which uses 16-bit words. Vim can support all of these encodings, but always uses UTF-8 internally.

Vim has comprehensive UTF-8 support. It works well in:
- xterm with utf-8 support enabled
- Athena, Motif and GTK GUI
- MS-Windows GUI
- several other platforms

Double-width characters are supported. This works best with 'guifontwide' or 'guifontset'. When using only 'guifont' the wide characters are drawn in the normal width and a space to fill the gap. Note that the 'guifontset' option is no longer relevant in the GTK+ 2 GUI.

bom-bytes

When reading a file a BOM (Byte Order Mark) can be used to recognize the Unicode encoding:

```
        EF BB BF     utf-8
        FE FF        utf-16 big endian
        FF FE        utf-16 little endian
        00 00 FE FF  utf-32 big endian
        FF FE 00 00  utf-32 little endian
```

Utf-8 is the recommended encoding. Note that it's difficult to tell utf-16 and utf-32 apart. Utf-16 is often used on MS-Windows, utf-32 is not widespread as file format.

mbyte-combining *mbyte-composing*

```
A composing or combining character is used to change the meaning of the
character before it.  The combining characters are drawn on top of the
preceding character.
Up to two combining characters can be used by default.  This can be changed
with the 'maxcombine' option.
When editing text a composing character is mostly considered part of the
preceding character.  For example "x" will delete a character and its
following composing characters by default.
If the 'delcombine' option is on, then pressing 'x' will delete the combining
characters, one at a time, then the base character.  But when inserting, you
type the first character and the following composing characters separately,
after which they will be joined.  The "r" command will not allow you to type a
combining character, because it doesn't know one is coming.  Use "R" instead.

Bytes which are not part of a valid UTF-8 byte sequence are handled like a
single character and displayed as <xx>, where "xx" is the hex value of the
byte.

Overlong sequences are not handled specially and displayed like a valid
character.  However, search patterns may not match on an overlong sequence.
(an overlong sequence is where more bytes are used than required for the
character.)  An exception is NUL (zero) which is displayed as "<00>".

In the file and buffer the full range of Unicode characters can be used (31
bits).  However, displaying only works for the characters present in the
selected font.

Useful commands:
- "ga" shows the decimal, hexadecimal and octal value of the character under
  the cursor.  If there are composing characters these are shown too.  (If the
  message is truncated, use ":messages").
- "g8" shows the bytes used in a UTF-8 character, also the composing
  characters, as hex numbers.
- ":set encoding=utf-8 fileencodings=" forces using UTF-8 for all files.  The
  default is to use the current locale for 'encoding' and set 'fileencodings'
  to automatically detect the encoding of a file.


STARTING VIM

If your current locale is in an utf-8 encoding, Vim will automatically start
in utf-8 mode.

If you are using another locale: >

        set encoding=utf-8

You might also want to select the font used for the menus.  Unfortunately this
doesn't always work.  See the system specific remarks below, and 'langmenu'.


USING UTF-8 IN X-Windows                                *utf-8-in-xwindows*

Note: This section does not apply to the GTK+ 2 GUI.

You need to specify a font to be used.  For double-wide characters another
font is required, which is exactly twice as wide.  There are three ways to do
this:

1. Set 'guifont' and let Vim find a matching 'guifontwide'
2. Set 'guifont' and 'guifontwide'
3. Set 'guifontset'
```

```
See the documentation for each option for details.  Example: >

   :set guifont=-misc-fixed-medium-r-normal--15-140-75-75-c-90-iso10646-1

You might also want to set the font used for the menus.  This only works for
Motif.  Use the ":hi Menu font={fontname}" command for this. |:highlight|


TYPING UTF-8                                            *utf-8-typing*

If you are using X-Windows, you should find an input method that supports
utf-8.

If your system does not provide support for typing utf-8, you can use the
'keymap' feature.  This allows writing a keymap file, which defines a utf-8
character as a sequence of ASCII characters.  See |mbyte-keymap|.

Another method is to set the current locale to the language you want to use
and for which you have a XIM available.  Then set 'termencoding' to that
language and Vim will convert the typed characters to 'encoding' for you.

If everything else fails, you can type any character as four hex bytes: >

        CTRL-V u 1234

"1234" is interpreted as a hex number.  You must type four characters, prepend
a zero if necessary.


COMMAND ARGUMENTS                                       *utf-8-char-arg*

Commands like |f|, |F|, |t| and |r| take an argument of one character.  For
UTF-8 this argument may include one or two composing characters.  These need
to be produced together with the base character, Vim doesn't wait for the next
character to be typed to find out if it is a composing character or not.
Using 'keymap' or |:lmap| is a nice way to type these characters.

The commands that search for a character in a line handle composing characters
as follows.  When searching for a character without a composing character,
this will find matches in the text with or without composing characters.  When
searching for a character with a composing character, this will only find
matches with that composing character.  It was implemented this way, because
not everybody is able to type a composing character.


==============================================================================
11. Overview of options                                 *mbyte-options*

These options are relevant for editing multi-byte files.  Check the help in
options.txt for detailed information.

'encoding'      Encoding used for the keyboard and display.  It is also the
                default encoding for files.

'fileencoding'  Encoding of a file.  When it's different from 'encoding'
                conversion is done when reading or writing the file.

'fileencodings' List of possible encodings of a file.  When opening a file
                these will be tried and the first one that doesn't cause an
                error is used for 'fileencoding'.
```

```
'charconvert'   Expression used to convert files from one encoding to another.

'formatoptions' The 'm' flag can be included to have formatting break a line
                at a multibyte character of 256 or higher.  Thus is useful for
                languages where a sequence of characters can be broken
                anywhere.

'guifontset'    The list of font names used for a multi-byte encoding.  When
                this option is not empty, it replaces 'guifont'.

'keymap'        Specify the name of a keyboard mapping.

==============================================================================

Contributions specifically for the multi-byte features by:
        Chi-Deok Hwang <hwang@mizi.co.kr>
        SungHyun Nam <goweol@gmail.com>
        K.Nagano <nagano@atese.advantest.co.jp>
        Taro Muraoka  <koron@tka.att.ne.jp>
        Yasuhiro Matsumoto <mattn@mail.goo.ne.jp>

 vim:tw=78:ts=8:ft=help:norl:
*mlang.txt*     For Vim version 8.0.  Last change: 2017 Mar 04


                  VIM REFERENCE MANUAL    by Bram Moolenaar


Multi-language features                         *multilang* *multi-lang*

This is about using messages and menus in various languages.  For editing
multi-byte text see |multibyte|.

The basics are explained in the user manual: |usr_45.txt|.

1. Messages                     |multilang-messages|
2. Menus                        |multilang-menus|
3. Scripts                      |multilang-scripts|

Also see |help-translated| for multi-language help.

{Vi does not have any of these features}
{not available when compiled without the |+multi_lang| feature}

==============================================================================
1. Messages                                             *multilang-messages*

Vim picks up the locale from the environment.  In most cases this means Vim
will use the language that you prefer, unless it's not available.

To see a list of supported locale names on your system, look in one of these
directories (for Unix):
        /usr/lib/locale ~
        /usr/share/locale ~
Unfortunately, upper/lowercase differences matter.  Also watch out for the
use of "-" and "_".

                                            *:lan* *:lang* *:language* *E197*
:lan[guage]
:lan[guage] mes[sages]
:lan[guage] cty[pe]
:lan[guage] tim[e]
```

```
                        Print the current language (aka locale).
                        With the "messages" argument the language used for
                        messages is printed.  Technical: LC_MESSAGES.
                        With the "ctype" argument the language used for
                        character encoding is printed.  Technical: LC_CTYPE.
                        With the "time" argument the language used for
                        strftime() is printed.  Technical: LC_TIME.
                        Without argument all parts of the locale are printed
                        (this is system dependent).
                        The current language can also be obtained with the
                        |v:lang|, |v:ctype| and |v:lc_time| variables.

:lan[guage] {name}
:lan[guage] mes[sages] {name}
:lan[guage] cty[pe] {name}
:lan[guage] tim[e] {name}
                        Set the current language (aka locale) to {name}.
                        The locale {name} must be a valid locale on your
                        system.  Some systems accept aliases like "en" or
                        "en_US", but some only accept the full specification
                        like "en_US.ISO_8859-1".  On Unix systems you can use
                        this command to see what locales are supported: >
                                :!locale -a
<                       With the "messages" argument the language used for
                        messages is set.  This can be different when you want,
                        for example, English messages while editing Japanese
                        text.  This sets $LC_MESSAGES.
                        With the "ctype" argument the language used for
                        character encoding is set.  This affects the libraries
                        that Vim was linked with.  It's unusual to set this to
                        a different value from 'encoding' or "C".  This sets
                        $LC_CTYPE.
                        With the "time" argument the language used for time
                        and date messages is set.  This affects strftime().
                        This sets $LC_TIME.
                        Without an argument both are set, and additionally
                        $LANG is set.
                        When compiled with the |+float| feature the LC_NUMERIC
                        value will always be set to "C", so that floating
                        point numbers use '.' as the decimal point.
                        This will make a difference for items that depend on
                        the language (some messages, time and date format).
                        Not fully supported on all systems
                        If this fails there will be an error message.  If it
                        succeeds there is no message.  Example: >
                                :language
                                Current language: C
                                :language de_DE.ISO_8859-1
                                :language mes
                                Current messages language: de_DE.ISO_8859-1
                                :lang mes en
<

MS-WINDOWS MESSAGE TRANSLATIONS                         *win32-gettext*

If you used the self-installing .exe file, message translations should work
already.  Otherwise get the libintl.dll file if you don't have it yet:

        http://sourceforge.net/projects/gettext
Or:
        https://mlocati.github.io/gettext-iconv-windows/
```

This also contains tools xgettext, msgformat and others.

libintl.dll should be placed in same directory with (g)vim.exe, or some place where PATH environment value describe. Vim also finds libintl-8.dll. Message files (vim.mo) have to be placed in "$VIMRUNTIME/lang/xx/LC_MESSAGES", where "xx" is the abbreviation of the language (mostly two letters).

If you write your own translations you need to generate the .po file and convert it to a .mo file. You need to get the source distribution and read the file "src/po/README.txt".

To overrule the automatic choice of the language, set the $LANG variable to the language of your choice. use "en" to disable translations. >

```
  :let $LANG = 'ja'
```

(text for Windows by Muraoka Taro)

==
2. Menus   *multilang-menus*

See |45.2| for the basics, esp. using 'langmenu'.

Note that if changes have been made to the menus after the translation was done, some of the menus may be shown in English. Please try contacting the maintainer of the translation and ask him to update it. You can find the name and e-mail address of the translator in "$VIMRUNTIME/lang/menu_<lang>.vim".

To set the font (or fontset) to use for the menus, use the |:highlight| command. Example: >

```
        :highlight Menu font=k12,r12
```

ALIAS LOCALE NAMES

Unfortunately, the locale names are different on various systems, even though they are for the same language and encoding. If you do not get the menu translations you expected, check the output of this command: >

```
        echo v:lang
```

Now check the "$VIMRUNTIME/lang" directory for menu translation files that use a similar language. A difference in a "-" being a "_" already causes a file not to be found! Another common difference to watch out for is "iso8859-1" versus "iso_8859-1". Fortunately Vim makes all names lowercase, thus you don't have to worry about case differences. Spaces are changed to underscores, to avoid having to escape them.

If you find a menu translation file for your language with a different name, create a file in your own runtime directory to load that one. The name of that file could be: >

```
        ~/.vim/lang/menu_<v:lang>.vim
```

Check the 'runtimepath' option for directories which are searched. In that file put a command to load the menu file with the other name: >

```
        runtime lang/menu_<other_lang>.vim
```

```
TRANSLATING MENUS

If you want to do your own translations, you can use the |:menutrans| command,
explained below.  It is recommended to put the translations for one language
in a Vim script.  For a language that has no translation yet, please consider
becoming the maintainer and make your translations available to all Vim users.
Send an e-mail to the Vim maintainer <maintainer@vim.org>.

                                        *:menut* *:menutrans* *:menutranslate*
:menut[ranslate] clear
                        Clear all menu translations.

:menut[ranslate] {english} {mylang}
                        Translate menu name {english} to {mylang}.  All
                        special characters like "&" and "<Tab>" need to be
                        included.  Spaces and dots need to be escaped with a
                        backslash, just like in other |:menu| commands.
                        Case in {english} is ignored.

See the $VIMRUNTIME/lang directory for examples.

To try out your translations you first have to remove all menus.  This is how
you can do it without restarting Vim: >
        :source $VIMRUNTIME/delmenu.vim
        :source <your-new-menu-file>
        :source $VIMRUNTIME/menu.vim

Each part of a menu path is translated separately.  The result is that when
"Help" is translated to "Hilfe" and "Overview" to "\DCberblick" then
"Help.Overview" will be translated to "Hilfe.\DCberblick".

==============================================================================
3. Scripts                                              *multilang-scripts*

In Vim scripts you can use the |v:lang| variable to get the current language
(locale).  The default value is "C" or comes from the $LANG environment
variable.

The following example shows how this variable is used in a simple way, to make
a message adapt to language preferences of the user, >

        :if v:lang =~ "de_DE"
        :  echo "Guten Morgen"
        :else
        :  echo "Good morning"
        :endif
<

 vim:tw=78:sw=4:ts=8:ft=help:norl:
*arabic.txt*    For Vim version 8.0.  Last change: 2010 Nov 13


                  VIM REFERENCE MANUAL    by Nadim Shaikli


Arabic Language support (options & mappings) for Vim            *Arabic*

{Vi does not have any of these commands}

                                                                *E800*
In order to use right-to-left and Arabic mapping support, it is
necessary to compile Vim with the |+arabic| feature.
```

```
These functions have been created by Nadim Shaikli <nadim-at-arabeyes.org>

It is best to view this file with these settings within Vim's GUI: >

        :set encoding=utf-8
        :set arabicshape


Introduction
------------
Arabic is a rather demanding language in which a number of special
features are required.  Characters are right-to-left oriented and
ought to appear as such on the screen (i.e. from right to left).
Arabic also requires shaping of its characters, meaning the same
character has a different visual form based on its relative location
within a word (initial, medial, final or stand-alone).  Arabic also
requires two different forms of combining and the ability, in
certain instances, to either superimpose up to two characters on top
of another (composing) or the actual substitution of two characters
into one (combining).  Lastly, to display Arabic properly one will
require not only ISO-8859-6 (U+0600-U+06FF) fonts, but will also
require Presentation Form-B (U+FE70-U+FEFF) fonts both of which are
subsets within a so-called ISO-10646-1 font.

The commands, prompts and help files are not in Arabic, therefore
the user interface remains the standard Vi interface.


Highlights
----------
o  Editing left-to-right files as in the original Vim hasn't changed.

o  Viewing and editing files in right-to-left windows.   File
   orientation is per window, so it is possible to view the same
   file in right-to-left and left-to-right modes, simultaneously.

o  No special terminal with right-to-left capabilities is required.
   The right-to-left changes are completely hardware independent.
   Only Arabic fonts are necessary.

o  Compatible with the original Vim.   Almost all features work in
   right-to-left mode (there are liable to be bugs).

o  Changing keyboard mapping and reverse insert modes using a single
   command.

o  Toggling complete Arabic support via a single command.

o  While in Arabic mode, numbers are entered from left to right.  Upon
   entering a none number character, that character will be inserted
   just into the left of the last number.

o  Arabic keymapping on the command line in reverse insert mode.

o  Proper Bidirectional functionality is possible given Vim is
   started within a Bidi capable terminal emulator.


Arabic Fonts                                           *arabicfonts*
------------
```

```
Vim requires monospaced fonts of which there are many out there.
Arabic requires ISO-8859-6 as well as Presentation Form-B fonts
(without Form-B, Arabic will _NOT_ be usable).  It is highly
recommended that users search for so-called 'ISO-10646-1' fonts.
Do an Internet search or check www.arabeyes.org for further
info on where to attain the necessary Arabic fonts.


Font Installation
-----------------

o  Installation of fonts for X Window systems (Unix/Linux)

   Depending on your system, copy your_ARABIC_FONT file into a
   directory of your choice.  Change to the directory containing
   the Arabic fonts and execute the following commands:

     %  mkfontdir
     %  xset +fp path_name_of_arabic_fonts_directory


Usage
-----
Prior to the actual usage of Arabic within Vim, a number of settings
need to be accounted for and invoked.

o  Setting the Arabic fonts

   +  For Vim GUI set the 'guifont' to your_ARABIC_FONT.  This is done
      by entering the following command in the Vim window.
>
                :set guifont=your_ARABIC_FONT
<
      NOTE: the string 'your_ARABIC_FONT' is used to denote a complete
            font name akin to that used in Linux/Unix systems.
            (e.g. -misc-fixed-medium-r-normal--20-200-75-75-c-100-iso10646-1)

      You can append the 'guifont' set command to your .vimrc file
      in order to get the same above noted results.  In other words,
      you can include ':set guifont=your_ARABIC_FONT' to your .vimrc
      file.

   +  Under the X Window environment, you can also start Vim with
      '-fn your_ARABIC_FONT' option.

o  Setting the appropriate character Encoding
   To enable the correct Arabic encoding the following command needs
   to be appended,
>
                :set encoding=utf-8
<
   to your .vimrc file (entering the command manually into you Vim
   window is highly discouraged).  In short, include ':set
   encoding=utf-8' to your .vimrc file.

   Attempts to use Arabic without UTF-8 will result the following
   warning message,

                                                          *W17*  >
     Arabic requires UTF-8, do ':set encoding=utf-8'

o  Enable Arabic settings [short-cut]
```

```
   In order to simplify and streamline things, you can either invoke
   Vim with the command-line option,

     % vim -A my_utf8_arabic_file ...

   or enable 'arabic' via the following command within Vim
>
                :set arabic
<
   The two above noted possible invocations are the preferred manner
   in which users are instructed to proceed.  Barring an enabled 'termbidi'
   setting, both command options:

     1. set the appropriate keymap
     2. enable the deletion of a single combined pair character
     3. enable rightleft    mode
     4. enable rightleftcmd mode (affecting the command-line)
     5. enable arabicshape  mode (do visual character alterations)

   You may also append the command to your .vimrc file and simply
   include ':set arabic' to it.

   You are also capable of disabling Arabic support via
>
                :set noarabic
<
   which resets everything that the command had enabled without touching
   the global settings as they could affect other possible open buffers.
   In short the 'noarabic' command,

     1. resets to the alternate keymap
     2. disables the deletion of a single combined pair character
     3. disables rightleft mode

   NOTE: the 'arabic' command takes into consideration 'termbidi' for
         possible external bi-directional (bidi) support from the
         terminal ("mlterm" for instance offers such support).
         'termbidi', if available, is superior to rightleft support
         and its support is preferred due to its level of offerings.
         'arabic' when 'termbidi' is enabled only sets the keymap.

   If, on the other hand, you'd like to be verbose and explicit and
   are opting not to use the 'arabic' short-cut command, here's what
   is needed (i.e. if you use ':set arabic' you can skip this section) -

   +  Arabic Keymapping Activation

      To activate the Arabic keymap (i.e. to remap your English/Latin
      keyboard to look-n-feel like a standard Arabic one), set the
      'keymap' command to "arabic".  This is done by entering
>
                :set keymap=arabic
<
      in your Vim window.  You can also append the 'keymap' set command to
      your .vimrc file.  In other words, you can include ':set keymap=arabic'
      to your .vimrc file.

      To turn toggle (or switch) your keymapping between Arabic and the
      default mapping (English), it is advised that users use the 'CTRL-^'
      key press while in insert (or add/replace) mode.  The command-line
      will display your current mapping by displaying an "Arabic" string
      next to your insertion mode (e.g. -- INSERT Arabic --) indicating
      your current keymap.

   +  Arabic deletion of a combined pair character

      By default Vim has the 'delcombine' option disabled.  This option
      allows the deletion of ALEF in a LAM_ALEF (LAA) combined character
      and still retain the LAM (i.e. it reverts to treating the combined
      character as its natural two characters form -- this also pertains
      to harakat and their combined forms).  You can enable this option
      by entering
>
                :set delcombine
<
      in our Vim window.  You can also append the 'delcombine' set command
      to your .vimrc file.  In other words, you can include ':set delcombine'
      to your .vimrc file.

   +  Arabic right-to-left Mode

      By default Vim starts in Left-to-right mode.  'rightleft' is the
      command that allows one to alter a window's orientation - that can
      be accomplished via,

      - Toggling between left-to-right and right-to-left modes is
        accomplished through ':set rightleft' and ':set norightleft'.

      - While in Left-to-right mode, enter ':set rl' in the command line
        ('rl' is the abbreviation for rightleft).

      - Put the ':set rl' line in your '.vimrc' file to start Vim in
        right-to-left mode permanently.

   +  Arabic right-to-left command-line Mode

      For certain commands the editing can be done in right-to-left mode.
      Currently this is only applicable to search commands.

      This is controlled with the 'rightleftcmd' option.  The default is
      "search", which means that windows in which 'rightleft' is set will
      edit search commands in right-left mode.  To disable this behavior,
>
                :set rightleftcmd=
<
      To enable right-left editing of search commands again,
>
                :set rightleftcmd&
<
   +  Arabic Shaping Mode

      To activate the required visual characters alterations (shaping,
      composing, combining) which the Arabic language requires, enable
      the 'arabicshape' command.  This is done by entering
>
                :set arabicshape
<
      in our Vim window.  You can also append the 'arabicshape' set
      command to your .vimrc file.  In other words, you can include
      ':set arabicshape' to your .vimrc file.


Keymap/Keyboard                                         *arabickeymap*
```

```
---------------

The character/letter encoding used in Vim is the standard UTF-8.
It is widely discouraged that any other encoding be used or even
attempted.

Note: UTF-8 is an all encompassing encoding and as such is
      the only supported (and encouraged) encoding with
      regard to Arabic (all other proprietary encodings
      should be discouraged and frowned upon).

o  Keyboard

   +  CTRL-^ in insert/replace mode toggles between Arabic/Latin mode

   +  Keyboard mapping is based on the Microsoft's Arabic keymap (the
      de facto standard in the Arab world):

  +---------------------------------------------------------------------+
  |!   |@   |#   |$   |%   |^   |&   |*   |(   |)   |_   |+   ||   |~  ّ |
  |1 ١ |2 ٢ |3 ٣ |4 ٤ |5 ٥ |6 ٦ |7 ٧ |8 ٨ |9 ٩ |0 ٠ |-   |=   |\   |` ذ |
  +---------------------------------------------------------------------+
       |Q  َ |W  ً |E  ُ |R  ٌ |T لإ |Y إ |U ` |I ÷ |O x |P ؛ |{ < |} > |
       |q ض |w ص |e ث |r ق |t ف |y غ |u ع |i ه |o خ |p ح |[ ج |] د |
       +-----------------------------------------------------------+
         |A  ِ |S  ٍ |D [ |F ] |G لأ |H أ |J ـ |K ، |L / |:   |"   |
         |a ش |s س |d ي |f ب |g ل |h ا |j ت |k ن |l م |; ك |' ط |
         +------------------------------------------------------+
           |Z ~ |X  ْ |C { |V } |B لآ |N آ |M ' |< , |> . |? ؟ |
           |z ئ |x ء |c ؤ |v ر |b لا |n ى |m ة |, و |. ز |/ ظ |
           +-------------------------------------------------+

Restrictions
------------

o  Vim in its GUI form does not currently support Bi-directionality
   (i.e. the ability to see both Arabic and Latin intermixed within
   the same line).


Known Bugs
----------

There is one known minor bug,

 1. If you insert a haraka (e.g. Fatha (U+064E)) after a LAM (U+0644)
    and then insert an ALEF (U+0627), the appropriate combining will
    not happen due to the sandwiched haraka resulting in something
    that will NOT be displayed correctly.

    WORK-AROUND: Don't include harakats between LAM and ALEF combos.
                 In general, don't anticipate to see correct visual
                 representation with regard to harakats and LAM+ALEF
                 combined characters (even those entered after both
                 characters).  The problem noted is strictly a visual
                 one, meaning saving such a file will contain all the
                 appropriate info/encodings - nothing is lost.

No other bugs are known to exist.

 vim:tw=78:ts=8:ft=help:norl:
*farsi.txt*     For Vim version 8.0.  Last change: 2015 Aug 29
```

```
                  VIM REFERENCE MANUAL    by Mortaza Ghassab Shiran


Right to Left and Farsi Mapping for Vim         *farsi* *Farsi*

{Vi does not have any of these commands}

                                                *E27*
In order to use right-to-left and Farsi mapping support, it is necessary to
compile Vim with the |+farsi| feature.

These functions have been made by Mortaza G. Shiran <shiran@jps.net>


Introduction
------------
In right-to-left oriented files the characters appear on the screen from right
to left.  This kind of file is most useful when writing Farsi documents,
composing faxes or writing Farsi memos.

The commands, prompts and help files are not in Farsi, therefore the user
interface remains the standard Vi interface.


Highlights
----------
o  Editing left-to-right files as in the original Vim, no change.

o  Viewing and editing files in right-to-left windows.   File orientation is
   per window, so it is possible to view the same file in right-to-left and
   left-to-right modes, simultaneously.

o  Compatibility to the original Vim.   Almost all features work in
   right-to-left mode (see bugs below).

o  Changing keyboard mapping and reverse insert modes using a single
   command.

o  Backing from reverse insert mode to the correct place in the file
   (if possible).

o  While in Farsi mode, numbers are entered from left to right.  Upon entering
   a none number character, that character will be inserted just into the
   left of the last number.

o  No special terminal with right-to-left capabilities is required.  The
   right-to-left changes are completely hardware independent.  Only
   Farsi font is necessary.

o  Farsi keymapping on the command line in reverse insert mode.

o  Toggling between left-to-right and right-to-left via F8 function key.

o  Toggling between Farsi ISIR-3342 standard encoding and Vim Farsi via F9
   function key.  Since this makes sense only for the text written in
   right-to-left mode, this function is also supported only in right-to-left
   mode.

Farsi Fonts                                     *farsi-fonts*
-----------
```

```
The following files are found in the subdirectories of the '$VIM/farsi/fonts'
directory:

   +  far-a01.pcf    X Windows fonts for Unix including Linux systems
   +  far-a01.bf     X Windows fonts for SunOS
   +  far-a01.f16    a screen fonts for Unix including Linux systems
   +  far-a01.fon    a monospaced fonts for Windows NT/95/98
   +  far-a01.com    a screen fonts for DOS


Font Installation
-----------------

o  Installation of fonts for MS Window systems (NT/95/98)

   From 'Control Panel' folder, start the 'Fonts' program.  Then from 'file'
   menu item select 'Install New Fonts ...'.  Browse and select the
   'far-a01.fon', then follow the installation guide.
   NOTE: several people have reported that this does not work.  The solution
   is unknown.

o  Installation of fonts for X Window systems (Unix/Linux)

   Depending on your system, copy far-a01.pcf.Z or far-a01.pcf.gz into a
   directory of your choice.  Change to the directory containing the Farsi
   fonts and execute the following commands:

   >  mkfontdir
   >  xset +fp path_name_of_farsi_fonts_directory

o  Installation of fonts for X Window systems (SunOS)

   Copy far-a01.bf font into a directory of your choice.
   Change to the directory containing the far-a01.fb fonts and
   execute the following commands:

   >  fldfamily
   >  xset +fp path_name_of_fonts_directory

o  Installation of ASCII screen fonts (Unix/Linux)

   For Linux system, copy the far-a01.f16 fonts into /usr/lib/kbd/consolefonts
   directory and execute the setfont program as "setfont far-a01.f16".  For
   other systems (e.g. SCO Unix), please refer to the fonts installation
   section of your system administration manuals.

o  Installation of ASCII screen fonts (DOS)

   After system power on, prior to the first use of Vim, upload the Farsi
   fonts by executing the far-a01.com font uploading program.


Usage
-----
Prior to starting Vim, the environment in which Vim can run in Farsi mode,
must be set.  In addition to installation of Farsi fonts, following points
refer to some of the system environments, which you may need to set:
Key code mapping, loading graphic card in ASCII screen mode, setting the IO
driver in 8 bit clean mode ... .

o  Setting the Farsi fonts
```

```
   +  For Vim GUI set the 'guifont' to far-a01.  This is done by entering
      ':set guifont=far-a01' in the Vim window.

      You can have 'guifont' set to far-a01 by Vim during the Vim startup
      by appending the ':set guifont=far-a01' into your .vimrc file
      (in case of NT/95/98 platforms _vimrc).

      Under the X Window environment, you can also start Vim with the
      '-fn far-a01' option.

   +  For Vim within a xterm, start a xterm with the Farsi fonts (e.g.
      kterm -fn far-a01).  Then start Vim inside the kterm.

   +  For Vim under DOS, prior to the first usage of Vim, upload the Farsi
      fonts by executing the far-a01.com fonts uploading program.

o  Farsi Keymapping Activation

   To activate the Farsi keymapping, set either 'altkeymap' or 'fkmap'.
   This is done by entering ':set akm' or ':set fk' in the Vim window.
   You can have 'altkeymap' or 'fkmap' set as default by appending ':set akm'
   or ':set fk' in your .vimrc file or _vimrc in case of NT/95/98 platforms.

   To turn off the Farsi keymapping as a default second language keymapping,
   reset the 'altkeymap' by entering ':set noakm'.

o  right-to-left Farsi Mode

   By default Vim starts in Left-to-right mode.  Following are ways to change
   the window orientation:

   + Start Vim with the -F option (e.g. vim -F ...).

   + Use the F8 function key to toggle between left-to-right and right-to-left.

   + While in Left-to-right mode, enter 'set rl' in the command line ('rl' is
     the abbreviation for rightleft).

   + Put the 'set rl' line in your '.vimrc' file to start Vim in
     right-to-left mode permanently.

Encoding
--------

The letter encoding used is the Vim extended ISIR-3342 standard with a built
in function to convert between Vim extended ISIR-3342 and ISIR-3342 standard.

For document portability reasons, the letter encoding is kept the same across
different platforms (i.e. UNIX's, NT/95/98, MS DOS, ...).


o  Keyboard

   +  CTRL-_ in insert/replace modes toggles between Farsi(akm)/Latin
      mode as follows:

   +  CTRL-_ moves the cursor to the end of the typed text in edit mode.

   +  CTRL-_ in command mode only toggles keyboard mapping between Farsi(akm)/
      Latin.  The Farsi text is then entered in reverse insert mode.
```

+ F8 - Toggles between left-to-right and right-to-left.

+ F9 - Toggles the encoding between ISIR-3342 standard and Vim extended
  ISIR-3342 (supported only in right-to-left mode).

+ Keyboard mapping is based on the Iranian ISIRI-2901 standard.
  Following table shows the keyboard mapping while Farsi(akm) mode set:

```
-------------------------------------
`  1  2  3  4  5  6  7  8  9  0  -  =
\A2  \B1  \B2  \B3  \B4  \B5  \B6  \B7  \B8  \B9  \B0  \AD  \BD
-------------------------------------
~  !  @  #  $  %  ^  &  *  (  )  _  +
~  \A3  \A7  \AE  \A4  \A5  \AA  \AC  \E8  \A8  \A9  \E9  \AB
-------------------------------------
q  w  e  r  t  z  u  i  o  p  [  ]
\D3  \D2  \C6  \D9  \D8  \D5  \D6  \E0  \CA  \C9  \C7  \88
-------------------------------------
Q  W  E  R  T  Z  U  I  O  P  {  }
\F7  \F5  \F4  \F3  \F2  \FD  \F0  \F6  [  ]  {  }
-------------------------------------
a  s  d  f  g  h  j  k  l  ;  '  \
\D1  \D0  \E1  \C3  \DC  \C1  \C5  \DE  \DD  \DA  \DB  \EB
-------------------------------------
A  S  D  F  G  H  J  K  L  :  "  |
\F9  \FB\A0 \FE  \FA  \F8  \C0  \FC  \E6  \E7  \BA  \BB  \EA
-------------------------------------
<  y  x  c  v  b  n  m  ,  .  /
\BE  \D7  \D4  \CE  \CD  \CC  \CB  \C4  \DF  \A6  \AF
-------------------------------------
>  Y  X  C  V  B  N  M  <  >  ?
\BC  \F1  \D4  \CF  \CD  \A1  \CB  \C2  \BE  \BC  \BF
-------------------------------------
```

Note:

```
        \A1  stands for Farsi PSP (break without space)

        \A2  stands for Farsi PCN (for HAMZE attribute)
```

Restrictions
----

o In insert/replace mode and fkmap (Farsi mode) set, CTRL-B is not
  supported.

o If you change the character mapping between Latin/Farsi, the redo buffer
  will be reset (emptied). That is, redo is valid and will function (using
  '.') only within the mode you are in.

o While numbers are entered in Farsi mode, the redo buffer will be reset
  (emptied). That is, you cannot redo the last changes (using '.') after
  entering numbers.

o While in left-to-right mode and Farsi mode set, CTRL-R is not supported.

o While in right-to-left mode, the search on 'Latin' pattern does not work,
  except if you enter the Latin search pattern in reverse.

o In command mode there is no support for entering numbers from left
  to right and also for the sake of flexibility the keymapping logic is
  restricted.

```
o  Under the X Window environment, if you want to run Vim within a xterm
   terminal emulator and Farsi mode set, you need to have an ANSI compatible
   xterm terminal emulator.  This is because the letter codes above 128 decimal
   have certain meanings in the standard xterm terminal emulator.

   Note: Under X Window environment, Vim GUI works fine in Farsi mode.
         This eliminates the need of any xterm terminal emulator.


Bugs
----
While in insert/replace and Farsi mode set, if you repeatedly change the
cursor position (via cursor movement) and enter new text and then try to undo
the last change, the undo will lag one change behind.  But as you continue to
undo, you will reach the original line of text.  You can also use U to undo all
changes made in the current line.

For more information about the bugs refer to rileft.txt.

 vim:tw=78:ts=8:ft=help:norl:
*hebrew.txt*    For Vim version 8.0.  Last change: 2007 Jun 14


           VIM REFERENCE MANUAL    by Ron Aaron (and Avner Lottem)


Hebrew Language support (options & mapping) for Vim             *hebrew*

The supporting 'rightleft' functionality was originally created by Avner
Lottem. <alottem at gmail dot com>  Ron Aaron <ron at ronware dot org> is
currently helping support these features.

{Vi does not have any of these commands}

All this is only available when the |+rightleft| feature was enabled at
compile time.


Introduction
------------
Hebrew-specific options are 'hkmap', 'hkmapp' 'keymap'=hebrew and 'aleph'.
Hebrew-useful options are 'delcombine', 'allowrevins', 'revins', 'rightleft'
and 'rightleftcmd'.

The 'rightleft' mode reverses the display order, so characters are displayed
from right to left instead of the usual left to right.  This is useful
primarily when editing Hebrew or other Middle-Eastern languages.
See |rileft.txt| for further details.

Details
--------------
+  Options:
   +  'rightleft' ('rl') sets window orientation to right-to-left.  This means
      that the logical text 'ABC' will be displayed as 'CBA', and will start
      drawing at the right edge of the window, not the left edge.
   +  'hkmap' ('hk') sets keyboard mapping to Hebrew, in insert/replace modes.
   +  'aleph' ('al'), numeric, holds the decimal code of Aleph, for keyboard
      mapping.
   +  'hkmapp' ('hkp') sets keyboard mapping to 'phonetic hebrew'

   NOTE: these three ('hkmap', 'hkmapp' and 'aleph') are obsolete.  You should
         use ":set keymap=hebrewp" instead.
```

   + 'delcombine' ('deco'), boolean, if editing UTF-8 encoded Hebrew, allows
     one to remove the niqud or te`amim by pressing 'x' on a character (with
     associated niqud).

   + 'rightleftcmd' ('rlc') makes the command-prompt for searches show up on
     the right side. It only takes effect if the window is 'rightleft'.

+ Encoding:
   + Under Unix, ISO 8859-8 encoding (Hebrew letters codes: 224-250).
   + Under MS DOS, PC encoding (Hebrew letters codes: 128-154).
     These are defaults, that can be overridden using the 'aleph' option.
   + You should prefer using UTF8, as it supports the combining-characters
     ('deco' does nothing if UTF8 encoding is not active).

+ Vim arguments:
   + 'vim -H file' starts editing a Hebrew file, i.e. 'rightleft' and 'hkmap'
     are set.

+ Keyboard:
   + The 'allowrevins' option enables the CTRL-_ command in Insert mode and
     in Command-line mode.

   + CTRL-_ in insert/replace modes toggles 'revins' and 'hkmap' as follows:

     When in rightleft window, 'revins' and 'nohkmap' are toggled, since
     English will likely be inserted in this case.

     When in norightleft window, 'revins' 'hkmap' are toggled, since Hebrew
     will likely be inserted in this case.

     CTRL-_ moves the cursor to the end of the typed text.

   + CTRL-_ in command mode only toggles keyboard mapping (see Bugs below).
     This setting is independent of 'hkmap' option, which only applies to
     insert/replace mode.

     Note: On some keyboards, CTRL-_ is mapped to CTRL-?.

   + Keyboard mapping while 'hkmap' is set (standard Israeli keyboard):

```
  q w e r t y u i o p
  / ' ק ר א ט ו ן ם פ

   a s d f g h j k l ; '
   ש ד ג כ ע י ח ל ך ף ,

    z x c v b n m , . /
    ז ס ב ה נ מ צ ת ץ .
```

     This is also the keymap when 'keymap=hebrew' is set. The advantage of
     'keymap' is that it works properly when using UTF8, e.g. it inserts the
     correct characters; 'hkmap' does not. The 'keymap' keyboard can also
     insert niqud and te`amim. To see what those mappings are, look at the
     keymap file 'hebrew.vim' etc.

Typing backwards

If the 'revins' (reverse insert) option is set, inserting happens backwards.
This can be used to type Hebrew. When inserting characters the cursor is not
moved and the text moves rightwards. A <BS> deletes the character under the

```
cursor.  CTRL-W and CTRL-U also work in the opposite direction.  <BS>, CTRL-W
and CTRL-U do not stop at the start of insert or end of line, no matter how
the 'backspace' option is set.

There is no reverse replace mode (yet).

If the 'showmode' option is set, "-- REVERSE INSERT --" will be shown in the
status line when reverse Insert mode is active.

When the 'allowrevins' option is set, reverse Insert mode can be also entered
via CTRL-_, which has some extra functionality: First, keyboard mapping is
changed according to the window orientation -- if in a left-to-right window,
'revins' is used to enter Hebrew text, so the keyboard changes to Hebrew
('hkmap' is set); if in a right-to-left window, 'revins' is used to enter
English text, so the keyboard changes to English ('hkmap' is reset).  Second,
when exiting 'revins' via CTRL-_, the cursor moves to the end of the typed
text (if possible).


Pasting when in a rightleft window
----------------------------------
When cutting text with the mouse and pasting it in a rightleft window
the text will be reversed, because the characters come from the cut buffer
from the left to the right, while inserted in the file from the right to
the left.   In order to avoid it, toggle 'revins' (by typing CTRL-? or CTRL-_)
before pasting.


Hebrew characters and the 'isprint' variable
--------------------------------------------
Sometimes Hebrew character codes are in the non-printable range defined by
the 'isprint' variable.  For example in the Linux console, the Hebrew font
encoding starts from 128, while the default 'isprint' variable is @,161-255.
The result is that all Hebrew characters are displayed as ~x.  To solve this
problem, set isprint=@,128-255.


 vim:tw=78:ts=8:ft=help:norl:
*russian.txt*   For Vim version 8.0.  Last change: 2006 Apr 24


                  VIM REFERENCE MANUAL    by Vassily Ragosin


Russian language localization and support in Vim          *russian* *Russian*

1. Introduction                         |russian-intro|
2. Russian keymaps                      |russian-keymap|
3. Localization                         |russian-l18n|
4. Known issues                         |russian-issues|

==============================================================================
1. Introduction                                             *russian-intro*

Russian language is supported perfectly well in Vim.  You can type and view
Russian text just as any other, without the need to tweak the settings.

==============================================================================
2. Russian keymaps                                         *russian-keymap*

To switch between languages you can use your system native keyboard switcher,
or use one of the Russian keymaps, included in the Vim distribution.  For
```

```
example,
>
    :set keymap=russian-jcukenwin
<
In the latter case, you can switch between languages even if you do not have
system Russian keyboard or independently from a system-wide keyboard settings.
See 'keymap'.  You can also map a key to switch between keyboards, if you
choose the latter option.  See |:map|.

For your convenience, to avoid switching between keyboards, when you need to
enter Normal mode command, you can also set 'langmap' option:
>
    :set langmap=ФИСВУАПРШОЛДЬТЩЗЙКЫЕГМЦЧНЯ;ABCDEFGHIJKLMNOPQRSTUVWXYZ,
    фисвуапршолдьтщзйкыегмцчня;abcdefghijklmnopqrstuvwxyz

This is in utf-8, you cannot read this if your 'encoding' is not utf-8.
You have to type this command in one line, it is wrapped for the sake of
readability.

==============================================================================
3. Localization                                                *russian-l18n*

If you wish to use messages, help files, menus and other items translated to
Russian, you will need to install the RuVim Language Pack, available in
different codepages from

    http://www.sourceforge.net/projects/ruvim/

Make sure that your Vim is at least 6.2.506 and use ruvim 0.5 or later for
automatic installs.  Vim also needs to be compiled with |+gettext| feature for
user interface items translations to work.

After downloading an archive from RuVim project, unpack it into your
$VIMRUNTIME directory.  We recommend using UTF-8 archive, if your version of
Vim is compiled with |+multi_byte| feature enabled.

In order to use the Russian documentation, make sure you have set the
'helplang' option to "ru".

==============================================================================
4. Known issues                                              *russian-issues*

-- If you are using Russian message translations in Win32 console, then
   you may see the output produced by "vim --help", "vim --version" commands
   and Win32 console window title appearing in a wrong codepage.  This problem
   is related to a bug in GNU gettext library and may be fixed in the future
   releases of gettext.

==============================================================================
 vim:tw=78:ts=8:ft=help:norl:
*ft_ada.txt*    For Vim version 8.0.  Last change: 2010 Jul 20


                    ADA FILE TYPE PLUG-INS REFERENCE MANUAL~

ADA                                                                 *ada.vim*

1.  Syntax Highlighting                     |ft-ada-syntax|
2.  File type Plug-in                       |ft-ada-plugin|
3.  Omni Completion                         |ft-ada-omni|
    3.1 Omni Completion with "gnat xref"        |gnat-xref|
    3.2 Omni Completion with "ctags"            |ada-ctags|
==============================================================================
1. Syntax Highlighting ~
                                                               *ft-ada-syntax*
```

This mode is designed for the 2005 edition of Ada ("Ada 2005"), which includes support for objected-programming, protected types, and so on. It handles code written for the original Ada language ("Ada83", "Ada87", "Ada95") as well, though code which uses Ada 2005-only keywords will be wrongly colored (such code should be fixed anyway). For more information about Ada, see http://www.adapower.com.

The Ada mode handles a number of situations cleanly.

For example, it knows that the "-" in "-5" is a number, but the same character in "A-5" is an operator. Normally, a "with" or "use" clause referencing another compilation unit is coloured the same way as C's "#include" is coloured. If you have "Conditional" or "Repeat" groups coloured differently, then "end if" and "end loop" will be coloured as part of those respective groups.

You can set these to different colours using vim's "highlight" command (e.g., to change how loops are displayed, enter the command ":hi Repeat" followed by the colour specification; on simple terminals the colour specification ctermfg=White often shows well).

There are several options you can select in this Ada mode. See |ft-ada-options| for a complete list.

To enable them, assign a value to the option. For example, to turn one on:

```
 >
    > let g:ada_standard_types = 1
>
```

To disable them use ":unlet". Example:

```
>
    > unlet g:ada_standard_types
```

You can just use ":" and type these into the command line to set these temporarily before loading an Ada file. You can make these option settings permanent by adding the "let" command(s), without a colon, to your "~/.vimrc" file.

Even on a slow (90Mhz) PC this mode works quickly, but if you find the performance unacceptable, turn on |g:ada_withuse_ordinary|.

Syntax folding instructions (|fold-syntax|) are added when |g:ada_folding| is set.

```
==============================================================================
2. File type Plug-in ~
                                               *ft-ada-indent* *ft-ada-plugin*
```

The Ada plug-in provides support for:

```
 - auto indenting       (|indent.txt|)
 - insert completion    (|i_CTRL-N|)
 - user completion      (|i_CTRL-X_CTRL-U|)
 - tag searches         (|tagsrch.txt|)
 - Quick Fix            (|quickfix.txt|)
 - backspace handling   (|'backspace'|)
 - comment handling     (|'comments'|, |'commentstring'|)

The plug-in only activates the features of the Ada mode whenever an Ada
file is opened and adds Ada related entries to the main and pop-up menu.

==============================================================================
3. Omni Completion ~
                                                                *ft-ada-omni*

The Ada omni-completions (|i_CTRL-X_CTRL-O|) uses tags database created either
by "gnat xref -v" or the "exuberant Ctags (http://ctags.sourceforge.net).  The
complete function will automatically detect which tool was used to create the
tags file.

------------------------------------------------------------------------------
3.1 Omni Completion with "gnat xref" ~
                                                                  *gnat-xref*

GNAT XREF uses the compiler internal information (ali-files) to produce the
tags file. This has the advantage to be 100% correct and the option of deep
nested analysis. However the code must compile, the generator is quite
slow and the created tags file contains only the basic Ctags information for
each entry - not enough for some of the more advanced Vim code browser
plug-ins.

NOTE: "gnat xref -v" is very tricky to use as it has almost no diagnostic
       output - If nothing is printed then usually the parameters are wrong.
       Here some important tips:

1)  You need to compile your code first and use the "-aO" option to point to
    your .ali files.
2)  "gnat xref -v ../Include/adacl.ads" won't work - use  the "gnat xref -v
    -aI../Include adacl.ads" instead.
3)  "gnat xref -v -aI../Include *.ad?" won't work - use "cd ../Include" and
    then "gnat xref -v *.ad?"
4)  Project manager support is completely broken - don't even try "gnat xref
    -Padacl.gpr".
5)  Vim is faster when the tags file is sorted - use "sort --unique
    --ignore-case --output=tags tags" .
6)  Remember to insert "!_TAG_FILE_SORTED 2 %sort ui" as first line to mark
    the file assorted.

------------------------------------------------------------------------------
3.2 Omni Completion with "ctags"~
                                                                  *ada-ctags*

Exuberant Ctags uses its own multi-language code parser. The parser is quite
fast, produces a lot of extra information (hence the name "Exuberant Ctags")
and can run on files which currently do not compile.

There are also lots of other Vim-tools which use exuberant Ctags.

You will need to install a version of the Exuberant Ctags which has Ada
support patched in. Such a version is available from the GNU Ada Project
(http://gnuada.sourceforge.net).
```

```
The Ada parser for Exuberant Ctags is fairly new - don't expect complete
support yet.

==============================================================================
4. Compiler Support ~
                                                                *ada-compiler*

The Ada mode supports more than one Ada compiler and will automatically load the
compiler set in |g:ada_default_compiler| whenever an Ada source is opened. The
provided compiler plug-ins are split into the actual compiler plug-in and a
collection of support functions and variables. This allows the easy
development of specialized compiler plug-ins fine tuned to your development
environment.

------------------------------------------------------------------------------
4.1 GNAT ~
                                                               *compiler-gnat*

GNAT is the only free (beer and speech) Ada compiler available. There are
several versions available which differ in the licence terms used.

The GNAT compiler plug-in will perform a compile on pressing <F7> and then
immediately shows the result. You can set the project file to be used by
setting:
 >
 > call g:gnat.Set_Project_File ('my_project.gpr')

Setting a project file will also create a Vim session (|views-sessions|) so -
like with the GPS - opened files, window positions etc. will be remembered
separately for all projects.

                                                                *gnat_members*
GNAT OBJECT ~

                                                               *g:gnat.Make()*
g:gnat.Make()
                Calls |g:gnat.Make_Command| and displays the result inside a
               |quickfix| window.

                                                             *g:gnat.Pretty()*
g:gnat.Pretty()
                Calls |g:gnat.Pretty_Program|

                                                               *g:gnat.Find()*
g:gnat.Find()
                Calls |g:gnat.Find_Program|

                                                               *g:gnat.Tags()*
g:gnat.Tags()
                Calls |g:gnat.Tags_Command|

                                                   *g:gnat.Set_Project_File()*
g:gnat.Set_Project_File([{file}])
                Set gnat project file and load associated session.  An open
                project will be closed and the session written.  If called
                without file name the file selector opens for selection of a
                project file. If called with an empty string then the project
                and associated session are closed.

                                                         *g:gnat.Project_File*
g:gnat.Project_File     string
```

```
                Current project file.

                                                        *g:gnat.Make_Command*
g:gnat.Make_Command     string
                External command used for |g:gnat.Make()| (|'makeprg'|).

                                                      *g:gnat.Pretty_Program*
g:gnat.Pretty_Program   string
                External command used for |g:gnat.Pretty()|

                                                        *g:gnat.Find_Program*
g:gnat.Find_Program     string
                External command used for |g:gnat.Find()|

                                                        *g:gnat.Tags_Command*
g:gnat.Tags_Command     string
                External command used for |g:gnat.Tags()|

                                                        *g:gnat.Error_Format*
g:gnat.Error_Format     string
                Error format (|'errorformat'|)

------------------------------------------------------------------------------
4.2 Dec Ada ~
                                         *compiler-hpada* *compiler-decada*
                                     *compiler-vaxada* *compiler-compaqada*

Dec Ada (also known by - in chronological order - VAX Ada, Dec Ada, Compaq Ada
and HP Ada) is a fairly dated Ada 83 compiler. Support is basic: <F7> will
compile the current unit.

The Dec Ada compiler expects the package name and not the file name to be
passed as a parameter. The compiler plug-in supports the usual file name
convention to convert the file into a unit name. Both '-' and '__' are allowed
as separators.

                                                             *decada_members*
DEC ADA OBJECT ~

                                                            *g:decada.Make()*
g:decada.Make()         function
                Calls |g:decada.Make_Command| and displays the result inside a
                |quickfix| window.

                                                       *g:decada.Unit_Name()*
g:decada.Unit_Name()    function
                Get the Unit name for the current file.

                                                      *g:decada.Make_Command*
g:decada.Make_Command   string
                External command used for |g:decada.Make()| (|'makeprg'|).

                                                      *g:decada.Error_Format*
g:decada.Error_Format|  string
                Error format (|'errorformat'|).

==============================================================================
5. References ~
                                                              *ada-reference*

------------------------------------------------------------------------------
5.1 Options ~
```

```
                                                       *ft-ada-options*

                                                 *g:ada_standard_types*
g:ada_standard_types    bool (true when exists)
                Highlight types in package Standard (e.g., "Float").

                                                   *g:ada_space_errors*
                                           *g:ada_no_trail_space_error*
                                             *g:ada_no_tab_space_error*
                                                  *g:ada_all_tab_usage*
g:ada_space_errors       bool (true when exists)
                Highlight extraneous errors in spaces ...
                g:ada_no_trail_space_error
                    - but ignore trailing spaces at the end of a line
                g:ada_no_tab_space_error
                    - but ignore tabs after spaces
                g:ada_all_tab_usage
                    - highlight all tab use

                                                    *g:ada_line_errors*
g:ada_line_errors         bool (true when exists)
                Highlight lines which are too long. Note: This highlighting
                option is quite CPU intensive.

                                                  *g:ada_rainbow_color*
g:ada_rainbow_color       bool (true when exists)
                Use rainbow colours for '(' and ')'. You need the
                rainbow_parenthesis for this to work.

                                                        *g:ada_folding*
g:ada_folding             set ('sigpft')
                Use folding for Ada sources.
                    's':    activate syntax folding on load
                        'p':    fold packages
                        'f':    fold functions and procedures
                        't':    fold types
                        'c':    fold conditionals
                    'g':    activate gnat pretty print folding on load
                        'i':    lone 'is' folded with line above
                        'b':    lone 'begin' folded with line above
                        'p':    lone 'private' folded with line above
                        'x':    lone 'exception' folded with line above
                    'i':    activate indent folding on load

                Note: Syntax folding is in an early (unusable) stage and
                      indent or gnat pretty folding is suggested.

                For gnat pretty folding to work the following settings are
                suggested: -cl3 -M79 -c2 -c3 -c4 -A1 -A2 -A3 -A4 -A5

                For indent folding to work the following settings are
                suggested: shiftwidth=3 softtabstop=3

                                                         *g:ada_abbrev*
g:ada_abbrev              bool (true when exists)
                Add some abbreviations. This feature is more or less superseded
                by the various completion methods.

                                              *g:ada_withuse_ordinary*
g:ada_withuse_ordinary    bool (true when exists)
                Show "with" and "use" as ordinary keywords (when used to
                reference other compilation units they're normally highlighted
                specially).

                                                       *g:ada_begin_preproc*
g:ada_begin_preproc       bool (true when exists)
                Show all begin-like keywords using the colouring of C
                preprocessor commands.

                                                  *g:ada_omni_with_keywords*
g:ada_omni_with_keywords
                Add Keywords, Pragmas, Attributes to omni-completions
                (|compl-omni|). Note: You can always complete then with user
                completion (|i_CTRL-X_CTRL-U|).

                                                    *g:ada_extended_tagging*
g:ada_extended_tagging    enum ('jump', 'list')
                use extended tagging, two options are available
                    'jump': use tjump to jump.
                    'list': add tags quick fix list.
                Normal tagging does not support function or operator
                overloading as these features are not available in C and
                tagging was originally developed for C.

                                                 *g:ada_extended_completion*
g:ada_extended_completion
                Uses extended completion for <C-N> and <C-R> completions
                (|i_CTRL-N|). In this mode the '.' is used as part of the
                identifier so that 'Object.Method' or 'Package.Procedure' are
                completed together.

                                                     *g:ada_gnat_extensions*
g:ada_gnat_extensions     bool (true when exists)
                 Support GNAT extensions.

                                              *g:ada_with_gnat_project_files*
g:ada_with_gnat_project_files    bool (true when exists)
                 Add gnat project file keywords and Attributes.

                                                    *g:ada_default_compiler*
g:ada_default_compiler    string
                set default compiler. Currently supported are 'gnat' and
                'decada'.

An "exists" type is a boolean considered true when the variable is defined and
false when the variable is undefined. The value to which the variable is set
makes no difference.

------------------------------------------------------------------------------
5.2 Commands ~
                                                            *ft-ada-commands*

:AdaRainbow                                                     *:AdaRainbow*
                Toggles rainbow colour (|g:ada_rainbow_color|) mode for
                '(' and ')'.

:AdaLines                                                         *:AdaLines*
                Toggles line error (|g:ada_line_errors|) display.

:AdaSpaces                                                       *:AdaSpaces*
                Toggles space error (|g:ada_space_errors|) display.

:AdaTagDir                                                       *:AdaTagDir*
                Creates tags file for the directory of the current file.
```

```
:AdaTagFile                                                    *:AdaTagFile*
                Creates tags file for the current file.

:AdaTypes                                                        *:AdaTypes*
                Toggles standard types (|g:ada_standard_types|) colour.

:GnatFind                                                        *:GnatFind*
                Calls |g:gnat.Find()|

:GnatPretty                                                    *:GnatPretty*
                Calls |g:gnat.Pretty()|

:GnatTags                                                        *:GnatTags*
                Calls |g:gnat.Tags()|

------------------------------------------------------------------------------
5.3 Variables ~
                                                          *ft-ada-variables*

                                                                  *g:gnat*
g:gnat                      object
                Control object which manages GNAT compiles.  The object
                is created when the first Ada source code is loaded provided
                that |g:ada_default_compiler| is set to 'gnat'. See
                |gnat_members| for details.

                                                                *g:decada*
g:decada                      object
                Control object which manages Dec Ada compiles.  The object
                is created when the first Ada source code is loaded provided
                that |g:ada_default_compiler| is set to 'decada'. See
                |decada_members| for details.

------------------------------------------------------------------------------
5.4 Constants ~
                                                          *ft-ada-constants*

All constants are locked. See |:lockvar| for details.

                                                         *g:ada#WordRegex*
g:ada#WordRegex         string
                Regular expression to search for Ada words.

                                                      *g:ada#DotWordRegex*
g:ada#DotWordRegex      string
                Regular expression to search for Ada words separated by dots.

                                                           *g:ada#Comment*
g:ada#Comment           string
                Regular expression to search for Ada comments.

                                                          *g:ada#Keywords*
g:ada#Keywords          list of dictionaries
                List of keywords, attributes etc. pp. in the format used by
                omni completion. See |complete-items| for details.

                                                       *g:ada#Ctags_Kinds*
g:ada#Ctags_Kinds       dictionary of lists
                Dictionary of the various kinds of items which the Ada support
                for Ctags generates.
```

```
------------------------------------------------------------------------------
5.5 Functions ~
                                                          *ft-ada-functions*

ada#Word([{line}, {col}])                                       *ada#Word()*
                Return full name of Ada entity under the cursor (or at given
                line/column), stripping white space/newlines as necessary.

ada#List_Tag([{line}, {col}])                               *ada#Listtags()*
                List all occurrences of the Ada entity under the cursor (or at
                given line/column) inside the quick-fix window.

ada#Jump_Tag ({ident}, {mode})                              *ada#Jump_Tag()*
                List all occurrences of the Ada entity under the cursor (or at
                given line/column) in the tag jump list. Mode can either be
                'tjump' or 'stjump'.

ada#Create_Tags ({option})                               *ada#Create_Tags()*
                Creates tag file using Ctags. The option can either be 'file'
                for the current file, 'dir' for the directory of the current
                file or a file name.

gnat#Insert_Tags_Header()                        *gnat#Insert_Tags_Header()*
                Adds the tag file header (!_TAG_) information to the current
                file which are missing from the GNAT XREF output.

ada#Switch_Syntax_Option ({option})             *ada#Switch_Syntax_Option()*
                Toggles highlighting options on or off. Used for the Ada menu.

                                                                *gnat#New()*
gnat#New ()
                Create a new gnat object. See |g:gnat| for details.


==============================================================================
6. Extra Plugins ~
                                                          *ada-extra-plugins*

You can optionally install the following extra plug-ins. They work well with
Ada and enhance the ability of the Ada mode:

backup.vim
        http://www.vim.org/scripts/script.php?script_id=1537
        Keeps as many backups as you like so you don't have to.

rainbow_parenthsis.vim
        http://www.vim.org/scripts/script.php?script_id=1561
        Very helpful since Ada uses only '(' and ')'.

nerd_comments.vim
        http://www.vim.org/scripts/script.php?script_id=1218
        Excellent commenting and uncommenting support for almost any
        programming language.

matchit.vim
        http://www.vim.org/scripts/script.php?script_id=39
        '%' jumping for any language. The normal '%' jump only works for '{}'
        style languages. The Ada mode will set the needed search patterns.

taglist.vim
        http://www.vim.org/scripts/script.php?script_id=273
        Source code explorer sidebar. There is a patch for Ada available.
```

```
The GNU Ada Project distribution (http://gnuada.sourceforge.net) of Vim
contains all of the above.

==============================================================================
vim: textwidth=78 nowrap tabstop=8 shiftwidth=4 softtabstop=4 noexpandtab
vim: filetype=help
*ft_sql.txt*    For Vim version 8.0.  Last change: 2013 May 15

by David Fishburn

This is a filetype plugin to work with SQL files.

The Structured Query Language (SQL) is a standard which specifies statements
that allow a user to interact with a relational database.  Vim includes
features for navigation, indentation and syntax highlighting.

1. Navigation                                   |sql-navigation|
    1.1 Matchit                                 |sql-matchit|
    1.2 Text Object Motions                     |sql-object-motions|
    1.3 Predefined Object Motions               |sql-predefined-objects|
    1.4 Macros                                  |sql-macros|
2. SQL Dialects                                 |sql-dialects|
    2.1 SQLSetType                              |SQLSetType|
    2.2 SQLGetType                              |SQLGetType|
    2.3 SQL Dialect Default                     |sql-type-default|
3. Adding new SQL Dialects                      |sql-adding-dialects|
4. OMNI SQL Completion                          |sql-completion|
    4.1 Static mode                             |sql-completion-static|
    4.2 Dynamic mode                            |sql-completion-dynamic|
    4.3 Tutorial                                |sql-completion-tutorial|
        4.3.1 Complete Tables                   |sql-completion-tables|
        4.3.2 Complete Columns                  |sql-completion-columns|
        4.3.3 Complete Procedures               |sql-completion-procedures|
        4.3.4 Complete Views                    |sql-completion-views|
    4.4 Completion Customization                |sql-completion-customization|
    4.5 SQL Maps                                |sql-completion-maps|
    4.6 Using with other filetypes              |sql-completion-filetypes|

==============================================================================
1. Navigation                                   *sql-navigation*

The SQL ftplugin provides a number of options to assist with file
navigation.


1.1 Matchit                                     *sql-matchit*
-----------
The matchit plugin (http://www.vim.org/scripts/script.php?script_id=39)
provides many additional features and can be customized for different
languages.  The matchit plugin is configured by defining a local
buffer variable, b:match_words.  Pressing the % key while on various
keywords will move the cursor to its match.  For example, if the cursor
is on an "if", pressing % will cycle between the "else", "elseif" and
"end if" keywords.

The following keywords are supported: >
    if
    elseif | elsif
    else [if]
    end if
    [while condition] loop
        leave
        break
        continue
        exit
    end loop

    for
        leave
        break
        continue
        exit
    end loop

    do
        statements
    doend

    case
    when
    when
    default
    end case

    merge
    when not matched
    when matched

    create[ or replace] procedure|function|event
    returns


1.2 Text Object Motions                         *sql-object-motions*
-----------------------
Vim has a number of predefined keys for working with text |object-motions|.
This filetype plugin attempts to translate these keys to maps which make sense
for the SQL language.

The following |Normal| mode and |Visual| mode maps exist (when you edit a SQL
file): >
    ]]              move forward to the next 'begin'
    [[              move backwards to the previous 'begin'
    ][              move forward to the next 'end'
    []              move backwards to the previous 'end'


1.3 Predefined Object Motions                   *sql-predefined-objects*
-----------------------------
Most relational databases support various standard features, tables, indices,
triggers and stored procedures.  Each vendor also has a variety of proprietary
objects.  The next set of maps have been created to help move between these
objects.  Depends on which database vendor you are using, the list of objects
must be configurable.  The filetype plugin attempts to define many of the
standard objects, plus many additional ones.  In order to make this as
flexible as possible, you can override the list of objects from within your
|vimrc| with the following: >
    let g:ftplugin_sql_objects = 'function,procedure,event,table,trigger' .
                \ ',schema,service,publication,database,datatype,domain' .
                \ ',index,subscription,synchronization,view,variable'

The following |Normal| mode and |Visual| mode maps have been created which use
the above list: >
```

```
    ]}              move forward to the next 'create <object name>'
    [{              move backward to the previous 'create <object name>'

Repeatedly pressing ]} will cycle through each of these create statements: >
    create table t1 (
        ...
    );

    create procedure p1
    begin
        ...
    end;

    create index i1 on t1 (c1);

The default setting for g:ftplugin_sql_objects is: >
    let g:ftplugin_sql_objects = 'function,procedure,event,' .
                \ '\\(existing\\\\|global\\s\\+temporary\\s\\+\\)\\\{,1}' .
                \ 'table,trigger' .
                \ ',schema,service,publication,database,datatype,domain' .
                \ ',index,subscription,synchronization,view,variable'

The above will also handle these cases: >
    create table t1 (
        ...
    );
    create existing table t2 (
        ...
    );
    create global temporary table t3 (
        ...
    );

By default, the ftplugin only searches for CREATE statements.  You can also
override this via your |vimrc| with the following: >
    let g:ftplugin_sql_statements = 'create,alter'

The filetype plugin defines three types of comments: >
    1.  --
    2.  //
    3.  /*
         *
         */

The following |Normal| mode and |Visual| mode maps have been created to work
with comments: >
    ]"              move forward to the beginning of a comment
    ["              move forward to the end of a comment



1.4 Macros                                          *sql-macros*
----------
Vim's feature to find macro definitions, |'define'|, is supported using this
regular expression: >
    \c\<\(VARIABLE\|DECLARE\|IN\|OUT\|INOUT\)\>

This addresses the following code: >
    CREATE VARIABLE myVar1 INTEGER;

    CREATE PROCEDURE sp_test(
        IN myVar2 INTEGER,
        OUT myVar3 CHAR(30),
        INOUT myVar4 NUMERIC(20,0)
    )
    BEGIN
        DECLARE myVar5 INTEGER;

        SELECT c1, c2, c3
          INTO myVar2, myVar3, myVar4
          FROM T1
         WHERE c4 = myVar1;
    END;
```

Place your cursor on "myVar1" on this line: >

```
         WHERE c4 = myVar1;
                     ^
```

Press any of the following keys: >

```
    [d
    [D
    [CTRL-D
```

==

```
2. SQL Dialects                                 *sql-dialects* *sql-types*
                                                *sybase* *TSQL* *Transact-SQL*
                                                *sqlanywhere*
                                                *oracle* *plsql* *sqlj*
                                                *sqlserver*
                                                *mysql* *postgresql* *psql*
                                                *informix*
```

All relational databases support SQL. There is a portion of SQL that is portable across vendors (ex. CREATE TABLE, CREATE INDEX), but there is a great deal of vendor specific extensions to SQL. Oracle supports the "CREATE OR REPLACE" syntax, column defaults specified in the CREATE TABLE statement and the procedural language (for stored procedures and triggers).

The default Vim distribution ships with syntax highlighting based on Oracle's PL/SQL. The default SQL indent script works for Oracle and SQL Anywhere. The default filetype plugin works for all vendors and should remain vendor neutral, but extendable.

Vim currently has support for a variety of different vendors, currently this is via syntax scripts. Unfortunately, to flip between different syntax rules you must either create:

```
    1.  New filetypes
    2.  Custom autocmds
    3.  Manual steps / commands
```

The majority of people work with only one vendor's database product, it would be nice to specify a default in your |vimrc|.

```
2.1 SQLSetType                                  *sqlsettype* *SQLSetType*
--------------
```

For the people that work with many different databases, it is nice to be able to flip between the various vendors rules (indent, syntax) on a per buffer basis, at any time. The ftplugin/sql.vim file defines this function: >

```
    SQLSetType
```

Executing this function without any parameters will set the indent and syntax scripts back to their defaults, see |sql-type-default|. If you have turned off Vi's compatibility mode, |'compatible'|, you can use the <Tab> key to complete the optional parameter.

After typing the function name and a space, you can use the completion to supply a parameter. The function takes the name of the Vim script you want to source. Using the |cmdline-completion| feature, the SQLSetType function will search the |'runtimepath'| for all Vim scripts with a name containing 'sql'. This takes the guess work out of the spelling of the names. The following are examples: >

```
    :SQLSetType
    :SQLSetType sqloracle
    :SQLSetType sqlanywhere
    :SQLSetType sqlinformix
    :SQLSetType mysql
```

The easiest approach is to the use <Tab> character which will first complete the command name (SQLSetType), after a space and another <Tab>, display a list of available Vim script names: >

```
    :SQL<Tab><space><Tab>
```

2.2 SQLGetType   *sqlgettype* *SQLGetType*

At anytime you can determine which SQL dialect you are using by calling the SQLGetType command. The ftplugin/sql.vim file defines this function: >

```
    SQLGetType
```

This will echo: >

```
    Current SQL dialect in use:sqlanywhere
```

2.3 SQL Dialect Default   *sql-type-default*

As mentioned earlier, the default syntax rules for Vim is based on Oracle (PL/SQL). You can override this default by placing one of the following in your |vimrc|: >

```
    let g:sql_type_default = 'sqlanywhere'
    let g:sql_type_default = 'sqlinformix'
    let g:sql_type_default = 'mysql'
```

If you added the following to your |vimrc|: >

```
    let g:sql_type_default = 'sqlinformix'
```

The next time edit a SQL file the following scripts will be automatically loaded by Vim: >

```
    ftplugin/sql.vim
    syntax/sqlinformix.vim
    indent/sql.vim
>
```

Notice indent/sqlinformix.sql was not loaded. There is no indent file for Informix, Vim loads the default files if the specified files does not exist.

==

3. Adding new SQL Dialects   *sql-adding-dialects*

If you begin working with a SQL dialect which does not have any customizations available with the default Vim distribution you can check http://www.vim.org to see if any customization currently exist. If not, you can begin by cloning an existing script. Read |filetype-plugins| for more details.

```
To help identify these scripts, try to create the files with a "sql" prefix.
If you decide you wish to create customizations for the SQLite database, you
can create any of the following: >
    Unix
        ~/.vim/syntax/sqlite.vim
        ~/.vim/indent/sqlite.vim
    Windows
        $VIM/vimfiles/syntax/sqlite.vim
        $VIM/vimfiles/indent/sqlite.vim

No changes are necessary to the SQLSetType function.  It will automatically
pickup the new SQL files and load them when you issue the SQLSetType command.


==============================================================================
4. OMNI SQL Completion                          *sql-completion*
                                                *omni-sql-completion*

Vim 7 includes a code completion interface and functions which allows plugin
developers to build in code completion for any language.  Vim 7 includes
code completion for the SQL language.

There are two modes to the SQL completion plugin, static and dynamic.  The
static mode populates the popups with the data generated from current syntax
highlight rules.  The dynamic mode populates the popups with data retrieved
directly from a database.  This includes, table lists, column lists,
procedures names and more.

4.1 Static Mode                                 *sql-completion-static*
---------------
The static popups created contain items defined by the active syntax rules
while editing a file with a filetype of SQL.  The plugin defines (by default)
various maps to help the user refine the list of items to be displayed.
The defaults static maps are: >
    imap <buffer> <C-C>a <C-\><C-O>:call sqlcomplete#Map('syntax')<CR><C-X><C-O>
    imap <buffer> <C-C>k <C-\><C-O>:call sqlcomplete#Map('sqlKeyword')<CR><C-X><C-O>
    imap <buffer> <C-C>f <C-\><C-O>:call sqlcomplete#Map('sqlFunction')<CR><C-X><C-O>
    imap <buffer> <C-C>o <C-\><C-O>:call sqlcomplete#Map('sqlOption')<CR><C-X><C-O>
    imap <buffer> <C-C>T <C-\><C-O>:call sqlcomplete#Map('sqlType')<CR><C-X><C-O>
    imap <buffer> <C-C>s <C-\><C-O>:call sqlcomplete#Map('sqlStatement')<CR><C-X><C-O>

The use of "<C-C>" can be user chosen by using the following in your |.vimrc| as it
may not work properly on all platforms: >
    let g:ftplugin_sql_omni_key = '<C-C>'
>
The static maps (which are based on the syntax highlight groups) follow this
format: >
    imap <buffer> <C-C>k <C-\><C-O>:call sqlcomplete#Map('sqlKeyword')<CR><C-X><C-O>
    imap <buffer> <C-C>k <C-\><C-O>:call sqlcomplete#Map('sqlKeyword\w*')<CR><C-X><C-
O>

This command breaks down as: >
    imap                   - Create an insert map
    <buffer>               - Only for this buffer
    <C-C>k                 - Your choice of key map
    <C-\><C-O>             - Execute one command, return to Insert mode
    :call sqlcomplete#Map( - Allows the SQL completion plugin to perform some
                             housekeeping functions to allow it to be used in
                             conjunction with other completion plugins.
                             Indicate which item you want the SQL completion
                             plugin to complete.
                             In this case we are asking the plugin to display
                             items from the syntax highlight group
                             'sqlKeyword'.
                             You can view a list of highlight group names to
                             choose from by executing the
                                 :syntax list
                             command while editing a SQL file.
    'sqlKeyword'           - Display the items for the sqlKeyword highlight
                             group
    'sqlKeyword\w*'        - A second option available with Vim 7.4 which
                             uses a regular expression to determine which
                             syntax groups to use
    )<CR>                  - Execute the :let command
    <C-X><C-O>             - Trigger the standard omni completion key stroke.
                             Passing in 'sqlKeyword' instructs the SQL
                             completion plugin to populate the popup with
                             items from the sqlKeyword highlight group.  The
                             plugin will also cache this result until Vim is
                             restarted.  The syntax list is retrieved using
                             the syntaxcomplete plugin.

Using the 'syntax' keyword is a special case.  This instructs the
syntaxcomplete plugin to retrieve all syntax items.  So this will effectively
work for any of Vim's SQL syntax files.  At the time of writing this includes
10 different syntax files for the different dialects of SQL (see section 3
above, |sql-dialects|).

Here are some examples of the entries which are pulled from the syntax files: >
     All
         - Contains the contents of all syntax highlight groups
     Statements
         - Select, Insert, Update, Delete, Create, Alter, ...
     Functions
         - Min, Max, Trim, Round, Date, ...
     Keywords
         - Index, Database, Having, Group, With
     Options
         - Isolation_level, On_error, Qualify_owners, Fire_triggers, ...
     Types
         - Integer, Char, Varchar, Date, DateTime, Timestamp, ...


4.2 Dynamic Mode                                *sql-completion-dynamic*
----------------
Dynamic mode populates the popups with data directly from a database.  In
order for the dynamic feature to be enabled you must have the dbext.vim
plugin installed, (http://vim.sourceforge.net/script.php?script_id=356).

Dynamic mode is used by several features of the SQL completion plugin.
After installing the dbext plugin see the dbext-tutorial for additional
configuration and usage.  The dbext plugin allows the SQL completion plugin
to display a list of tables, procedures, views and columns. >
     Table List
         - All tables for all schema owners
     Procedure List
         - All stored procedures for all schema owners
     View List
         - All stored procedures for all schema owners
     Column List
         - For the selected table, the columns that are part of the table

To enable the popup, while in INSERT mode, use the following key combinations
for each group (where <C-C> means hold the CTRL key down while pressing
the space bar):
     Table List            - <C-C>t
                           - <C-X><C-O> (the default map assumes tables)
     Stored Procedure List - <C-C>p
     View List             - <C-C>v
     Column List           - <C-C>c

     Drilling In / Out     - When viewing a popup window displaying the list
                             of tables, you can press <Right>, this will
                             replace the table currently highlighted with
                             the column list for that table.
                           - When viewing a popup window displaying the list
                             of columns, you can press <Left>, this will
                             replace the column list with the list of tables.
                           - This allows you to quickly drill down into a
                             table to view its columns and back again.
                           - <Right> and <Left> can be also be chosen via
                             your |.vimrc| >
                                let g:ftplugin_sql_omni_key_right = '<Right>'
                                let g:ftplugin_sql_omni_key_left  = '<Left>'

The SQL completion plugin caches various lists that are displayed in
the popup window.  This makes the re-displaying of these lists very
fast.  If new tables or columns are added to the database it may become
necessary to clear the plugins cache.  The default map for this is: >
    imap <buffer> <C-C>R <C-\><C-O>:call sqlcomplete#Map('ResetCache')<CR><C-X><C-O>


4.3 SQL Tutorial                                *sql-completion-tutorial*
----------------

This tutorial is designed to take you through the common features of the SQL
completion plugin so that: >
     a) You gain familiarity with the plugin
     b) You are introduced to some of the more common features
     c) Show how to customize it to your preferences
     d) Demonstrate "Best of Use" of the plugin (easiest way to configure).

First, create a new buffer: >
     :e tutorial.sql


Static features
---------------
To take you through the various lists, simply enter insert mode, hit:
    <C-C>s   (show SQL statements)
At this point, you can page down through the list until you find "select".
If you are familiar with the item you are looking for, for example you know
the statement begins with the letter "s".  You can type ahead (without the
quotes) "se" then press:
    <C-Space>t
Assuming "select" is highlighted in the popup list press <Enter> to choose
the entry.  Now type:
    * fr<C-C>a (show all syntax items)
choose "from" from the popup list.

When writing stored procedures using the "type" list is useful.  It contains
a list of all the database supported types.  This may or may not be true
depending on the syntax file you are using.  The SQL Anywhere syntax file
(sqlanywhere.vim) has support for this: >
     BEGIN
        DECLARE customer_id <C-C>T <-- Choose a type from the list
```

Dynamic features

To take advantage of the dynamic features you must first install the
dbext.vim plugin (http://vim.sourceforge.net/script.php?script_id=356). It
also comes with a tutorial. From the SQL completion plugin's perspective,
the main feature dbext provides is a connection to a database. dbext
connection profiles are the most efficient mechanism to define connection
information. Once connections have been setup, the SQL completion plugin
uses the features of dbext in the background to populate the popups.

What follows assumes dbext.vim has been correctly configured, a simple test
is to run the command, :DBListTable. If a list of tables is shown, you know
dbext.vim is working as expected. If not, please consult the dbext.txt
documentation.

Assuming you have followed the dbext-tutorial you can press <C-C>t to
display a list of tables. There is a delay while dbext is creating the table
list. After the list is displayed press <C-W>. This will remove both the
popup window and the table name already chosen when the list became active. >

```
 4.3.1 Table Completion:                       *sql-completion-tables*
```

Press <C-C>t to display a list of tables from within the database you
have connected via the dbext plugin.
NOTE: All of the SQL completion popups support typing a prefix before pressing
the key map. This will limit the contents of the popup window to just items
beginning with those characters. >

```
 4.3.2 Column Completion:                      *sql-completion-columns*
```

The SQL completion plugin can also display a list of columns for particular
tables. The column completion is trigger via <C-C>c.

NOTE: The following example uses <Right> to trigger a column list while
the popup window is active.

Example of using column completion:
- Press <C-C>t again to display the list of tables.
- When the list is displayed in the completion window, press <Right>,
  this will replace the list of tables, with a list of columns for the
  table highlighted (after the same short delay).
- If you press <Left>, this will again replace the column list with the
  list of tables. This allows you to drill into tables and column lists
  very quickly.
- Press <Right> again while the same table is highlighted. You will
  notice there is no delay since the column list has been cached. If you
  change the schema of a cached table you can press <C-C>R, which
  clears the SQL completion cache.
- NOTE: <Right> and <Left> have been designed to work while the
  completion window is active. If the completion popup window is
  not active, a normal <Right> or <Left> will be executed.

Let's look at how we can build a SQL statement dynamically. A select statement
requires a list of columns. There are two ways to build a column list using
the SQL completion plugin. >

```
    One column at a time:
<       1. After typing SELECT press <C-C>t to display a list of tables.
        2. Choose a table from the list.
        3. Press <Right> to display a list of columns.
        4. Choose the column from the list and press enter.
        5. Enter a "," and press <C-C>c.  Generating a column list
           generally requires having the cursor on a table name.  The plugin
           uses this name to determine what table to retrieve the column list.
           In this step, since we are pressing <C-C>c without the cursor
           on a table name the column list displayed will be for the previous
           table.  Choose a different column and move on.
        6. Repeat step 5 as often as necessary. >
    All columns for a table:
<       1. After typing SELECT press <C-C>t to display a list of tables.
        2. Highlight the table you need the column list for.
        3. Press <Enter> to choose the table from the list.
        4. Press <C-C>l to request a comma separated list of all columns
           for this table.
        5. Based on the table name chosen in step 3, the plugin attempts to
           decide on a reasonable table alias.  You are then prompted to
           either accept of change the alias.  Press OK.
        6. The table name is replaced with the column list of the table is
           replaced with the comma separate list of columns with the alias
           prepended to each of the columns.
        7. Step 3 and 4 can be replaced by pressing <C-C>L, which has
           a <C-Y> embedded in the map to choose the currently highlighted
           table in the list.

There is a special provision when writing select statements.  Consider the
following statement: >
     select *
       from customer c,
            contact cn,
            department as dp,
            employee e,
            site_options so
      where c.

In INSERT mode after typing the final "c." which is an alias for the
"customer" table, you can press either <C-C>c or <C-X><C-O>.  This will
popup a list of columns for the customer table.  It does this by looking back
to the beginning of the select statement and finding a list of the tables
specified in the FROM clause.  In this case it notes that in the string
"customer c", "c" is an alias for the customer table.  The optional "AS"
keyword is also supported, "customer AS c". >


 4.3.3 Procedure Completion:                   *sql-completion-procedures*

Similar to the table list, <C-C>p, will display a list of stored
procedures stored within the database. >

 4.3.4 View Completion:                        *sql-completion-views*

Similar to the table list, <C-C>v, will display a list of views in the
database.


4.4 Completion Customization                   *sql-completion-customization*
----------------------------

The SQL completion plugin can be customized through various options set in
your |vimrc|: >
    omni_sql_no_default_maps
<       - Default: This variable is not defined
        - If this variable is defined, no maps are created for OMNI
          completion.  See |sql-completion-maps| for further discussion.
```

```
>
    omni_sql_use_tbl_alias
<       - Default: a
        - This setting is only used when generating a comma separated
          column list.  By default the map is <C-C>l.  When generating
          a column list, an alias can be prepended to the beginning of each
          column, for example:  e.emp_id, e.emp_name.  This option has three
          settings: >
                n - do not use an alias
                d - use the default (calculated) alias
                a - ask to confirm the alias name
<
          An alias is determined following a few rules:
               1.  If the table name has an '_', then use it as a separator: >
                   MY_TABLE_NAME --> MTN
                   my_table_name --> mtn
                   My_table_NAME --> MtN
<              2.  If the table name does NOT contain an '_', but DOES use
                   mixed case then the case is used as a separator: >
                   MyTableName --> MTN
<              3.  If the table name does NOT contain an '_', and does NOT
                   use mixed case then the first letter of the table is used: >
                   mytablename --> m
                   MYTABLENAME --> M

    omni_sql_ignorecase
<       - Default: Current setting for 'ignorecase'
        - Valid settings are 0 or 1.
        - When entering a few letters before initiating completion, the list
          will be filtered to display only the entries which begin with the
          list of characters.  When this option is set to 0, the list will be
          filtered using case sensitivity. >

    omni_sql_include_owner
<       - Default: 0, unless dbext.vim 3.00 has been installed
        - Valid settings are 0 or 1.
        - When completing tables, procedure or views and using dbext.vim 3.00
          or higher the list of objects will also include the owner name.
          When completing these objects and omni_sql_include_owner is enabled
          the owner name will be replaced. >

    omni_sql_precache_syntax_groups
<       - Default:
          ['syntax','sqlKeyword','sqlFunction','sqlOption','sqlType','sqlStatement']
        - sqlcomplete can be used in conjunction with other completion
          plugins.  This is outlined at |sql-completion-filetypes|.  When the
          filetype is changed temporarily to SQL, the sqlcompletion plugin
          will cache the syntax groups listed in the List specified in this
          option.
>

4.5 SQL Maps                                    *sql-completion-maps*
------------

The default SQL maps have been described in other sections of this document in
greater detail.  Here is a list of the maps with a brief description of each.

Static Maps
-----------
These are maps which use populate the completion list using Vim's syntax
highlighting rules. >
    <C-C>a
```

```
<       - Displays all SQL syntax items. >
    <C-C>k
<       - Displays all SQL syntax items defined as 'sqlKeyword'. >
    <C-C>f
<       - Displays all SQL syntax items defined as 'sqlFunction. >
    <C-C>o
<       - Displays all SQL syntax items defined as 'sqlOption'. >
    <C-C>T
<       - Displays all SQL syntax items defined as 'sqlType'. >
    <C-C>s
<       - Displays all SQL syntax items defined as 'sqlStatement'. >

Dynamic Maps
------------
These are maps which use populate the completion list using the dbext.vim
plugin. >
    <C-C>t
<       - Displays a list of tables. >
    <C-C>p
<       - Displays a list of procedures. >
    <C-C>v
<       - Displays a list of views. >
    <C-C>c
<       - Displays a list of columns for a specific table. >
    <C-C>l
<       - Displays a comma separated list of columns for a specific table. >
    <C-C>L
<       - Displays a comma separated list of columns for a specific table.
          This should only be used when the completion window is active. >
    <Right>
<       - Displays a list of columns for the table currently highlighted in
          the completion window.  <Right> is not recognized on most Unix
          systems, so this maps is only created on the Windows platform.
          If you would like the same feature on Unix, choose a different key
          and make the same map in your vimrc. >
    <Left>
<       - Displays the list of tables.
          <Left> is not recognized on most Unix systems, so this maps is
          only created on the Windows platform.  If you would like the same
          feature on Unix, choose a different key and make the same map in
          your vimrc. >
    <C-C>R
<       - This maps removes all cached items and forces the SQL completion
          to regenerate the list of items.

Customizing Maps
----------------
You can create as many additional key maps as you like.  Generally, the maps
will be specifying different syntax highlight groups.

If you do not wish the default maps created or the key choices do not work on
your platform (often a case on *nix) you define the following variable in
your |vimrc|: >
    let g:omni_sql_no_default_maps = 1

Do no edit ftplugin/sql.vim directly!  If you change this file your changes
will be over written on future updates.  Vim has a special directory structure
which allows you to make customizations without changing the files that are
included with the Vim distribution.  If you wish to customize the maps
create an after/ftplugin/sql.vim (see |after-directory|) and place the same
maps from the ftplugin/sql.vim in it using your own key strokes.  <C-C> was
chosen since it will work on both Windows and *nix platforms.  On the windows
```

```
platform you can also use <C-Space> or ALT keys.


4.6 Using with other filetypes                  *sql-completion-filetypes*
------------------------------

Many times SQL can be used with different filetypes.  For example Perl, Java,
PHP, Javascript can all interact with a database.  Often you need both the SQL
completion and the completion capabilities for the current language you are
editing.

This can be enabled easily with the following steps (assuming a Perl file): >
    1.  :e test.pl
    2.  :set filetype=sql
    3.  :set ft=perl

Step 1
------
Begins by editing a Perl file.  Vim automatically sets the filetype to
"perl".  By default, Vim runs the appropriate filetype file
ftplugin/perl.vim.  If you are using the syntax completion plugin by following
the directions at |ft-syntax-omni| then the |'omnifunc'| option has been set to
"syntax#Complete".  Pressing <C-X><C-O> will display the omni popup containing
the syntax items for Perl.

Step 2
------
Manually setting the filetype to 'sql' will also fire the appropriate filetype
files ftplugin/sql.vim.  This file will define a number of buffer specific
maps for SQL completion, see |sql-completion-maps|.  Now these maps have
been created and the SQL completion plugin has been initialized.  All SQL
syntax items have been cached in preparation.  The SQL filetype script detects
we are attempting to use two different completion plugins.  Since the SQL maps
begin with <C-C>, the maps will toggle the |'omnifunc'| when in use.  So you
can use <C-X><C-O> to continue using the completion for Perl (using the syntax
completion plugin) and <C-C> to use the SQL completion features.

Step 3
------
Setting the filetype back to Perl sets all the usual "perl" related items back
as they were.


vim:tw=78:ts=8:ft=help:norl:
*hangulin.txt*  For Vim version 8.0.  Last change: 2015 Nov 24


                  VIM REFERENCE MANUAL    by Chi-Deok Hwang and Sung-Hyun Nam


Introduction                                    *hangul*
------------
It is to input hangul, the Korean language, with Vim GUI version.
If you have a XIM program, you can use another |+xim| feature.
Basically, it is for anybody who has no XIM program.

Compile
-------
Next is a basic option.  You can add any other configure option. >

   ./configure --with-x --enable-multibyte --enable-hangulinput \
            --disable-xim
```

```
And you should check feature.h.  If |+hangul_input| feature is enabled
by configure, you can select more options such as keyboard type, 2 bulsik
or 3 bulsik.  You can find keywords like next in there. >

        #define HANGUL_DEFAULT_KEYBOARD 2
        #define ESC_CHG_TO_ENG_MODE
        /* #define X_LOCALE */

Environment variables
---------------------
You should set LANG variable to Korean locale such as ko, ko_KR.eucKR
or ko_KR.UTF-8.
If you set LC_ALL variable, it should be set to Korean locale also.

Vim resource
------------
You may want to set 'encoding' and 'fileencodings'.
Next are examples: >

        :set encoding=euc-kr
        :set encoding=utf-8
        :set fileencodings=ucs-bom,utf-8,cp949,euc-kr,latin1

Keyboard
--------
You can change keyboard type (2 bulsik or 3 bulsik) using VIM_KEYBOARD
or HANGUL_KEYBOARD_TYPE environment variables.  For sh, just do (2 bulsik): >

    export VIM_KEYBOARD="2"
or >
    export HANGUL_KEYBOARD_TYPE="2"

If both are set, VIM_KEYBOARD has higher priority.

Hangul Fonts
------------
If you use GTK version of gvim, you should set 'guifont' and 'guifontwide'.
For example: >
    set guifont=Courier\ 12
    set guifontwide=NanumGothicCoding\ 12

If you use Motif or Athena version of gvim, you should set 'guifontset' in
your vimrc.  You can set fontset in the .Xdefaults file.

$HOME/.gvimrc: >
    set guifontset=english_font,hangul_font

$HOME/.Xdefaults: >
    Vim.font: english_font

    ! Nexts are for hangul menu with Athena
    *international: True
    Vim*fontSet: english_font,hangul_font

    ! Nexts are for hangul menu with Motif
    *international: True
    Vim*fontList: english_font;hangul_font:

attention! the , (comma) or ; (semicolon)

And there should be no ':set guifont'.  If it exists, then gvim ignores
```

```
':set guifontset'.  It means Vim runs without fontset supporting.
So, you can see only English.  Hangul does not be correctly displayed.

After "fontset" feature is enabled, Vim does not allow using english
font only in "font" setting for syntax.
For example, if you use >
   :set guifontset=eng_font,your_font
in your .gvimrc, then you should do for syntax >
   :hi Comment guifg=Cyan font=another_eng_font,another_your_font
If you just do >
   :hi Comment font=another_eng_font
then you can see a error message.  Be careful!

hangul_font width should be twice than english_font width.

Unsupported Feature
-------------------
We don't support Johab font.
We don't support Hanja input.
And We don't have any plan to support them.

If you really need such features, you can use console version of Vim with a
capable terminal emulator.

Bug or Comment
--------------
Send comments, patches and suggestions to:

                                    SungHyun Nam <goweol@gmail.com>
                                    Chi-Deok Hwang <...>

 vim:tw=78:ts=8:ft=help:norl:
*rileft.txt*    For Vim version 8.0.  Last change: 2006 Apr 24


                  VIM REFERENCE MANUAL    by Avner Lottem
                                          updated by Nadim Shaikli


Right to Left display mode for Vim                              *rileft*


These functions were originally created by Avner Lottem:
   E-mail: alottem@iil.intel.com
   Phone:  +972-4-8307322

{Vi does not have any of these commands}

                                                                *E26*
{only available when compiled with the |+rightleft| feature}


Introduction
------------
Some languages such as Arabic, Farsi, Hebrew (among others) require the
ability to display their text from right-to-left.  Files in those languages
are stored conventionally and the right-to-left requirement is only a
function of the display engine (per the Unicode specification).  In
right-to-left oriented files the characters appear on the screen from
right to left.

Bidirectionality (or bidi for short) is what Unicode offers as a full
```

```
solution to these languages.  Bidi offers the user the ability to view
both right-to-left as well as left-to-right text properly at the same time
within the same window.  Vim currently, due to simplicity, does not offer
bidi and is merely opting to present a functional means to display/enter/use
right-to-left languages.  An older hybrid solution in which direction is
encoded for every character (or group of characters) are not supported either
as this kind of support is out of the scope of a simple addition to an
existing editor (and it's not sanctioned by Unicode either).


Highlights
----------
o  Editing left-to-right files as in the original Vim, no change.

o  Viewing and editing files in right-to-left windows.  File orientation
   is per window, so it is possible to view the same file in right-to-left
   and left-to-right modes, simultaneously.  (Useful for editing mixed files
   in which both right-to-left and left-to-right text exist).

o  Compatibility to the original Vim.  Almost all features work in
   right-to-left mode (see Bugs below).

o  Backing from reverse insert mode to the correct place in the file
   (if possible).

o  No special terminal with right-to-left capabilities is required.  The
   right-to-left changes are completely hardware independent.

o  Many languages use and require right-to-left support.  These languages
   can quite easily be supported given the inclusion of their required
   keyboard mappings and some possible minor code change.  Some of the
   current supported languages include - |arabic.txt|, |farsi.txt| and
   |hebrew.txt|.


Of Interest...
--------------

o  Invocations
   -----------
   + 'rightleft' ('rl') sets window orientation to right-to-left.
   + 'delcombine' ('deco'), boolean, if editing UTF-8 encoded languages,
     allows one to remove a composing character which gets superimposed
     on those that proceeded them (some languages require this).
   + 'rightleftcmd' ('rlc') sets the command-line within certain modes
     (such as search) to be utilized in right-to-left orientation as well.

o  Typing backwards                                     *ins-reverse*
   ----------------
   In lieu of using full-fledged the 'rightleft' option, one can opt for
   reverse insertion.  When the 'revins' (reverse insert) option is set,
   inserting happens backwards.  This can be used to type right-to-left
   text.  When inserting characters the cursor is not moved and the text
   moves rightwards.  A <BS> deletes the character under the cursor.
   CTRL-W and CTRL-U also work in the opposite direction.  <BS>, CTRL-W
   and CTRL-U do not stop at the start of insert or end of line, no matter
   how the 'backspace' option is set.

   There is no reverse replace mode (yet).

   If the 'showmode' option is set, "-- REVERSE INSERT --" will be shown
   in the status line when reverse Insert mode is active.
```

```
o  Pasting when in a rightleft window
   ----------------------------------
   When cutting text with the mouse and pasting it in a rightleft window
   the text will be reversed, because the characters come from the cut buffer
   from the left to the right, while inserted in the file from the right to
   the left.   In order to avoid it, toggle 'revins' before pasting.


Bugs
----
o  Does not handle CTRL-A and CTRL-X commands (add and subtract) correctly
   when in rightleft window.

o  Does not support reverse insert and rightleft modes on the command-line.
   However, functionality of the editor is not reduced, because it is
   possible to enter mappings, abbreviations and searches typed from the
   left to the right on the command-line.

o  Somewhat slower in right-to-left mode, because right-to-left motion is
   emulated inside Vim, not by the controlling terminal.

o  When the Athena GUI is used, the bottom scrollbar works in the wrong
   direction.  This is difficult to fix.

o  When both 'rightleft' and 'revins' are on: 'textwidth' does not work.
   Lines do not wrap at all; you just get a single, long line.

o  There is no full bidirectionality (bidi) support.


 vim:tw=78:ts=8:ft=help:norl:
```