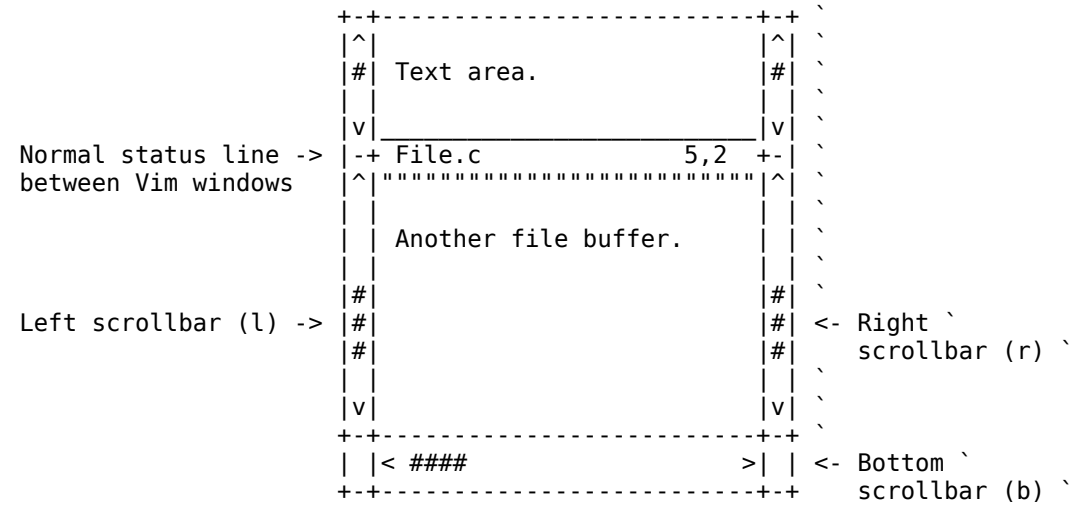
```
GUI ~
|gui.txt|   Graphical User Interface (GUI)
|gui_w32.txt|   Win32 GUI
|gui_x11.txt|   X11 GUI

==============================================================================
*gui.txt*       For Vim version 8.0.  Last change: 2017 Sep 23


                  VIM REFERENCE MANUAL    by Bram Moolenaar


Vim's Graphical User Interface                          *gui* *GUI*

1. Starting the GUI             |gui-start|
2. Scrollbars                   |gui-scrollbars|
3. Mouse Control                |gui-mouse|
4. Making GUI Selections        |gui-selections|
5. Menus                        |menus|
6. Extras                       |gui-extras|
7. Shell Commands               |gui-shell|

Other GUI documentation:
|gui_x11.txt|   For specific items of the X11 GUI.
|gui_w32.txt|   For specific items of the Win32 GUI.

{Vi does not have any of these commands}

==============================================================================
1. Starting the GUI                                *gui-start* *E229* *E233*

First you must make sure you actually have a version of Vim with the GUI code
included.  You can check this with the ":version" command, it says "with xxx
GUI", where "xxx" is X11-Motif, X11-Athena, Photon, GTK2, GTK3, etc., or
"MS-Windows 32 bit GUI version".

How to start the GUI depends on the system used.  Mostly you can run the
GUI version of Vim with:
    gvim [options] [files...]

The X11 version of Vim can run both in GUI and in non-GUI mode.  See
|gui-x11-start|.

                        *gui-init* *gvimrc* *.gvimrc* *_gvimrc* *$MYGVIMRC*
The gvimrc file is where GUI-specific startup commands should be placed.  It
is always sourced after the |vimrc| file.  If you have one then the $MYGVIMRC
environment variable has its name.

When the GUI starts up initializations are carried out, in this order:
- The 'term' option is set to "builtin_gui" and terminal options are reset to
  their default value for the GUI |terminal-options|.
- If the system menu file exists, it is sourced.  The name of this file is
  normally "$VIMRUNTIME/menu.vim".  You can check this with ":version".  Also
  see |$VIMRUNTIME|.  To skip loading the system menu include 'M' in
  'guioptions'.                         *buffers-menu* *no_buffers_menu*
  The system menu file includes a "Buffers" menu.  If you don't want this, set
  the "no_buffers_menu" variable in your .vimrc (not .gvimrc!): >
        :let no_buffers_menu = 1
<  NOTE: Switching on syntax highlighting also loads the menu file, thus
  disabling the Buffers menu must be done before ":syntax on".
  The path names are truncated to 35 characters.  You can truncate them at a
  different length, for example 50, like this: >
```

```
        :let bmenu_max_pathlen = 50
- If the "-U {gvimrc}" command-line option has been used when starting Vim,
  the {gvimrc} file will be read for initializations.  The following
  initializations are skipped.  When {gvimrc} is "NONE" no file will be read
  for initializations.
- For Unix and MS-Windows, if the system gvimrc exists, it is sourced.  The
  name of this file is normally "$VIM/gvimrc".  You can check this with
  ":version".  Also see |$VIM|.
- The following are tried, and only the first one that exists is used:
  - If the GVIMINIT environment variable exists and is not empty, it is
    executed as an Ex command.
  - If the user gvimrc file exists, it is sourced.  The name of this file is
    normally "$HOME/.gvimrc".  You can check this with ":version".
  - For Win32, $HOME is set by Vim if needed, see |$HOME-windows|.
  - When a "_gvimrc" file is not found, ".gvimrc" is tried too.  And vice
    versa.
  The name of the first file found is stored in $MYGVIMRC, unless it was
  already set.
- If the 'exrc' option is set (which is NOT the default) the file ./.gvimrc
  is sourced, if it exists and isn't the same file as the system or user
  gvimrc file.  If this file is not owned by you, some security restrictions
  apply.  When ".gvimrc" is not found, "_gvimrc" is tried too.  For Macintosh
  and DOS/Win32 "_gvimrc" is tried first.

NOTE: All but the first one are not carried out if Vim was started with
"-u NONE" or "-u DEFAULTS" and no "-U" argument was given, or when started
with "-U NONE".

All this happens AFTER the normal Vim initializations, like reading your
.vimrc file.  See |initialization|.
But the GUI window is only opened after all the initializations have been
carried out.  If you want some commands to be executed just after opening the
GUI window, use the |GUIEnter| autocommand event.  Example: >
        :autocmd GUIEnter * winpos 100 50

You can use the gvimrc files to set up your own customized menus (see |:menu|)
and initialize other things that you may want to set up differently from the
terminal version.

Recommended place for your personal GUI initializations:
        Unix                $HOME/.gvimrc or $HOME/.vim/gvimrc
        OS/2                $HOME/.gvimrc, $HOME/vimfiles/gvimrc
                            or $VIM/.gvimrc
        MS-DOS and Win32    $HOME/_gvimrc, $HOME/vimfiles/gvimrc
                            or $VIM/_gvimrc
        Amiga               s:.gvimrc, home:.gvimrc, home:vimfiles:gvimrc
                            or $VIM/.gvimrc

The personal initialization files are searched in the order specified above
and only the first one that is found is read.

There are a number of options which only have meaning in the GUI version of
Vim.  These are 'guicursor', 'guifont', 'guipty' and 'guioptions'.  They are
documented in |options.txt| with all the other options.

If using the Motif or Athena version of the GUI (but not for the GTK+ or
Win32 version), a number of X resources are available.  See |gui-resources|.

Another way to set the colors for different occasions is with highlight
groups.  The "Normal" group is used to set the background and foreground
colors.  Example (which looks nice): >
```

```
        :highlight Normal guibg=grey90

The "guibg" and "guifg" settings override the normal background and
foreground settings.  The other settings for the Normal highlight group are
not used.  Use the 'guifont' option to set the font.

Also check out the 'guicursor' option, to set the colors for the cursor in
various modes.

Vim tries to make the window fit on the screen when it starts up.  This avoids
that you can't see part of it.  On the X Window System this requires a bit of
guesswork.  You can change the height that is used for the window title and a
task bar with the 'guiheadroom' option.

                                                *:winp* *:winpos* *E188*
:winp[os]
                Display current position of the top left corner of the GUI vim
                window in pixels.  Does not work in all versions.
                Also see |getwinposx()| and |getwinposy()|.

:winp[os] {X} {Y}                                                       *E466*
                Put the GUI vim window at the given {X} and {Y} coordinates.
                The coordinates should specify the position in pixels of the
                top left corner of the window.  Does not work in all versions.
                Does work in an (new) xterm |xterm-color|.
                When the GUI window has not been opened yet, the values are
                remembered until the window is opened.  The position is
                adjusted to make the window fit on the screen (if possible).

                                                    *:win* *:winsize* *E465*
:win[size] {width} {height}
                Set the window height to {width} by {height} characters.
                Obsolete, use ":set lines=11 columns=22".
                If you get less lines than expected, check the 'guiheadroom'
                option.

If you are running the X Window System, you can get information about the
window Vim is running in with these commands: >
        :!xwininfo -id $WINDOWID
        :!xprop -id $WINDOWID
        :execute '!xwininfo -id ' . v:windowid
        :execute '!xprop -id ' . v:windowid
<
                                                        *gui-IME* *iBus*
Input methods for international characters in X that rely on the XIM
framework, most notably iBus, have been known to produce undesirable results
in gVim. These may include an inability to enter spaces, or long delays
between typing a character and it being recognized by the application.

One workaround that has been successful, for unknown reasons, is to prevent
gvim from forking into the background by starting it with the |-f| argument.

==============================================================================
2. Scrollbars                                           *gui-scrollbars*

There are vertical scrollbars and a horizontal scrollbar.  You may
configure which ones appear with the 'guioptions' option.

The interface looks like this (with ":set guioptions=mlrb"):

                      +------------------------------+ `
                      | File  Edit             Help | <- Menu bar (m) `
```



Any of the scrollbar or menu components may be turned off by not putting the
appropriate letter in the 'guioptions' string. The bottom scrollbar is
only useful when 'nowrap' is set.

VERTICAL SCROLLBARS *gui-vert-scroll*

Each Vim window has a scrollbar next to it which may be scrolled up and down
to move through the text in that buffer. The size of the scrollbar-thumb
indicates the fraction of the buffer which can be seen in the window.
When the scrollbar is dragged all the way down, the last line of the file
will appear in the top of the window.

If a window is shrunk to zero height (by the growth of another window) its
scrollbar disappears. It reappears when the window is restored.

If a window is vertically split, it will get a scrollbar when it is the
current window and when, taking the middle of the current window and drawing a
vertical line, this line goes through the window.
When there are scrollbars on both sides, and the middle of the current window
is on the left half, the right scrollbar column will contain scrollbars for
the rightmost windows. The same happens on the other side.

HORIZONTAL SCROLLBARS *gui-horiz-scroll*

The horizontal scrollbar (at the bottom of the Vim GUI) may be used to
scroll text sideways when the 'wrap' option is turned off. The
scrollbar-thumb size is such that the text of the longest visible line may be
scrolled as far as possible left and right. The cursor is moved when
necessary, it must remain on a visible character (unless 'virtualedit' is
set).

Computing the length of the longest visible line takes quite a bit of
computation, and it has to be done every time something changes. If this
takes too much time or you don't like the cursor jumping to another line,
include the 'h' flag in 'guioptions'. Then the scrolling is limited by the
text of the current cursor line.

athena-intellimouse
If you have an Intellimouse and an X server that supports using the wheel,
then you can use the wheel to scroll the text up and down in gvim. This works
with XFree86 4.0 and later, and with some older versions when you add patches.

```
See |scroll-mouse-wheel|.

For older versions of XFree86 you must patch your X server.  The following
page has a bit of information about using the Intellimouse on Linux as well as
links to the patches and X server binaries (may not have the one you need
though):
    http://www.inria.fr/koala/colas/mouse-wheel-scroll/

==============================================================================
3. Mouse Control                                        *gui-mouse*

The mouse only works if the appropriate flag in the 'mouse' option is set.
When the GUI is switched on, and 'mouse' wasn't set yet, the 'mouse' option is
automatically set to "a", enabling it for all modes except for the
|hit-enter| prompt.  If you don't want this, a good place to change the
'mouse' option is the "gvimrc" file.

Other options that are relevant:
'mousefocus'    window focus follows mouse pointer |gui-mouse-focus|
'mousemodel'    what mouse button does which action
'mousehide'     hide mouse pointer while typing text
'selectmode'    whether to start Select mode or Visual mode

A quick way to set these is with the ":behave" command.
                                                        *:behave* *:be*
:be[have] {model}       Set behavior for mouse and selection.  Valid
                        arguments are:
                           mswin        MS-Windows behavior
                           xterm        Xterm behavior

                        Using ":behave" changes these options:
                        option          mswin                   xterm   ~
                        'selectmode'    "mouse,key"             ""
                        'mousemodel'    "popup"                 "extend"
                        'keymodel'      "startsel,stopsel"      ""
                        'selection'     "exclusive"             "inclusive"

In the $VIMRUNTIME directory, there is a script called |mswin.vim|, which will
also map a few keys to the MS-Windows cut/copy/paste commands.  This is NOT
compatible, since it uses the CTRL-V, CTRL-X and CTRL-C keys.  If you don't
mind, use this command: >
        :so $VIMRUNTIME/mswin.vim

For scrolling with a wheel on a mouse, see |scroll-mouse-wheel|.


3.1 Moving Cursor with Mouse                            *gui-mouse-move*

Click the left mouse button somewhere in a text buffer where you want the
cursor to go, and it does!
This works in       when 'mouse' contains ~
Normal mode         'n' or 'a'
Visual mode         'v' or 'a'
Insert mode         'i' or 'a'

Select mode is handled like Visual mode.

You may use this with an operator such as 'd' to delete text from the current
cursor position to the position you point to with the mouse.  That is, you hit
'd' and then click the mouse somewhere.

                                                        *gui-mouse-focus*
```

The 'mousefocus' option can be set to make the keyboard focus follow the
mouse pointer. This means that the window where the mouse pointer is, is the
active window. Warning: this doesn't work very well when using a menu,
because the menu command will always be applied to the top window.

If you are on the ':' line (or '/' or '?'), then clicking the left or right
mouse button will position the cursor on the ':' line (if 'mouse' contains
'c', 'a' or 'A').

In any situation the middle mouse button may be clicked to paste the current
selection.

3.2 Selection with Mouse *gui-mouse-select*

The mouse can be used to start a selection. How depends on the 'mousemodel'
option:
'mousemodel' is "extend": use the right mouse button
'mousemodel' is "popup": use the left mouse button, while keeping the Shift
key pressed.

If there was no selection yet, this starts a selection from the old cursor
position to the position pointed to with the mouse. If there already is a
selection then the closest end will be extended.

If 'selectmode' contains "mouse", then the selection will be in Select mode.
This means that typing normal text will replace the selection. See
|Select-mode|. Otherwise, the selection will be in Visual mode.

Double clicking may be done to make the selection word-wise, triple clicking
makes it line-wise, and quadruple clicking makes it rectangular block-wise.

See |gui-selections| on how the selection is used.

3.3 Other Text Selection with Mouse *gui-mouse-modeless* *modeless-selection*

A different kind of selection is used when:
- in Command-line mode
- in the Command-line window and pointing in another window
- at the |hit-enter| prompt
- whenever the current mode is not in the 'mouse' option
- when holding the CTRL and SHIFT keys in the GUI

Since Vim continues like the selection isn't there, and there is no mode
associated with the selection, this is called modeless selection. Any text in
the Vim window can be selected. Select the text by pressing the left mouse
button at the start, drag to the end and release. To extend the selection,
use the right mouse button when 'mousemodel' is "extend", or the left mouse
button with the shift key pressed when 'mousemodel' is "popup".
The selection is removed when the selected text is scrolled or changed.

On the command line CTRL-Y can be used to copy the selection into the
clipboard. To do this from Insert mode, use CTRL-O : CTRL-Y <CR>. When
'guioptions' contains a or A (default on X11), the selection is automatically
copied to the "* register.

The middle mouse button can then paste the text. On non-X11 systems, you can
use CTRL-R +.

3.4 Using Mouse on Status Lines *gui-mouse-status*

Clicking the left or right mouse button on the status line below a Vim
window makes that window the current window. This actually happens on button
release (to be able to distinguish a click from a drag action).

With the left mouse button a status line can be dragged up and down, thus
resizing the windows above and below it. This does not change window focus.

The same can be used on the vertical separator: click to give the window left
of it focus, drag left and right to make windows wider and narrower.

3.5 Various Mouse Clicks *gui-mouse-various*

<S-LeftMouse>	Search forward for the word under the mouse click. When 'mousemodel' is "popup" this starts or extends a selection.
<S-RightMouse>	Search backward for the word under the mouse click.
<C-LeftMouse>	Jump to the tag name under the mouse click.
<C-RightMouse>	Jump back to position before the previous tag jump (same as "CTRL-T")

3.6 Mouse Mappings *gui-mouse-mapping*

The mouse events, complete with modifiers, may be mapped. Eg: >
```
   :map <S-LeftMouse>     <RightMouse>
   :map <S-LeftDrag>      <RightDrag>
   :map <S-LeftRelease>   <RightRelease>
   :map <2-S-LeftMouse>   <2-RightMouse>
   :map <2-S-LeftDrag>    <2-RightDrag>
   :map <2-S-LeftRelease> <2-RightRelease>
   :map <3-S-LeftMouse>   <3-RightMouse>
   :map <3-S-LeftDrag>    <3-RightDrag>
   :map <3-S-LeftRelease> <3-RightRelease>
   :map <4-S-LeftMouse>   <4-RightMouse>
   :map <4-S-LeftDrag>    <4-RightDrag>
   :map <4-S-LeftRelease> <4-RightRelease>
```
These mappings make selection work the way it probably should in a Motif
application, with shift-left mouse allowing for extending the visual area
rather than the right mouse button.

Mouse mapping with modifiers does not work for modeless selection.

3.7 Drag and drop *drag-n-drop*

You can drag and drop one or more files into the Vim window, where they will
be opened as if a |:drop| command was used.

If you hold down Shift while doing this, Vim changes to the first dropped
file's directory. If you hold Ctrl Vim will always split a new window for the
file. Otherwise it's only done if the current buffer has been changed.

You can also drop a directory on Vim. This starts the explorer plugin for
that directory (assuming it was enabled, otherwise you'll get an error
message). Keep Shift pressed to change to the directory instead.

If Vim happens to be editing a command line, the names of the dropped files
and directories will be inserted at the cursor. This allows you to use these
names with any Ex command. Special characters (space, tab, double quote and
'|'; backslash on non-MS-Windows systems) will be escaped.

==
4. Making GUI Selections *gui-selections*

quotestar
You may make selections with the mouse (see |gui-mouse-select|), or by using
Vim's Visual mode (see |v|). If 'a' is present in 'guioptions', then
whenever a selection is started (Visual or Select mode), or when the selection
is changed, Vim becomes the owner of the windowing system's primary selection
(on MS-Windows the |gui-clipboard| is used; under X11, the |x11-selection| is
used - you should read whichever of these is appropriate now).

clipboard
There is a special register for storing this selection, it is the "*
register. Nothing is put in here unless the information about what text is
selected is about to change (e.g. with a left mouse click somewhere), or when
another application wants to paste the selected text. Then the text is put
in the "* register. For example, to cut a line and make it the current
selection/put it on the clipboard: >

```
"*dd
```

Similarly, when you want to paste a selection from another application, e.g.,
by clicking the middle mouse button, the selection is put in the "* register
first, and then 'put' like any other register. For example, to put the
selection (contents of the clipboard): >

```
"*p
```

When using this register under X11, also see |x11-selection|. This also
explains the related "+ register.

Note that when pasting text from one Vim into another separate Vim, the type
of selection (character, line, or block) will also be copied. For other
applications the type is always character. However, if the text gets
transferred via the |x11-cut-buffer|, the selection type is ALWAYS lost.

When the "unnamed" string is included in the 'clipboard' option, the unnamed
register is the same as the "* register. Thus you can yank to and paste the
selection without prepending "* to commands.

==
5. Menus *menus*

For an introduction see |usr_42.txt| in the user manual.

5.1 Using Menus *using-menus*

Basically, menus can be used just like mappings. You can define your own
menus, as many as you like.
Long-time Vim users won't use menus much. But the power is in adding your own
menus and menu items. They are most useful for things that you can't remember
what the key sequence was.

For creating menus in a different language, see |:menutrans|.

menu.vim
The default menus are read from the file "$VIMRUNTIME/menu.vim". See
|$VIMRUNTIME| for where the path comes from. You can set up your own menus.
Starting off with the default set is a good idea. You can add more items, or,
if you don't like the defaults at all, start with removing all menus

|:unmenu-all|. You can also avoid the default menus being loaded by adding
this line to your .vimrc file (NOT your .gvimrc file!): >
        :let did_install_default_menus = 1
If you also want to avoid the Syntax menu: >
        :let did_install_syntax_menu = 1
The first item in the Syntax menu can be used to show all available filetypes
in the menu (which can take a bit of time to load). If you want to have all
filetypes already present at startup, add: >
        :let do_syntax_sel_menu = 1

<

console-menus
Although this documentation is in the GUI section, you can actually use menus
in console mode too. You will have to load |menu.vim| explicitly then, it is
not done by default. You can use the |:emenu| command and command-line
completion with 'wildmenu' to access the menu entries almost like a real menu
system. To do this, put these commands in your .vimrc file: >
        :source $VIMRUNTIME/menu.vim
        :set wildmenu
        :set cpo-=<
        :set wcm=<C-Z>
        :map <F4> :emenu <C-Z>
Pressing <F4> will start the menu. You can now use the cursor keys to select
a menu entry. Hit <Enter> to execute it. Hit <Esc> if you want to cancel.
This does require the |+menu| feature enabled at compile time.

tear-off-menus
GTK+ 2 and Motif support Tear-off menus. These are sort of sticky menus or
pop-up menus that are present all the time. If the resizing does not work
correctly, this may be caused by using something like "Vim*geometry" in the
defaults. Use "Vim.geometry" instead.

As to GTK+ 3, tear-off menus have been deprecated since GTK+ 3.4.
Accordingly, they are disabled if gvim is linked against GTK+ 3.4 or later.

The Win32 GUI version emulates Motif's tear-off menus. Actually, a Motif user
will spot the differences easily, but hopefully they're just as useful. You
can also use the |:tearoff| command together with |hidden-menus| to create
floating menus that do not appear on the main menu bar.

5.2 Creating New Menus *creating-menus*

:me *:menu* *:noreme* *:noremenu*
:am *:amenu* *:an*   *:anoremenu*
:nme *:nmenu* *:nnoreme* *:nnoremenu*
:ome *:omenu* *:onoreme* *:onoremenu*
:vme *:vmenu* *:vnoreme* *:vnoremenu*
:xme *:xmenu* *:xnoreme* *:xnoremenu*
:sme *:smenu* *:snoreme* *:snoremenu*
:ime *:imenu* *:inoreme* *:inoremenu*
:cme *:cmenu* *:cnoreme* *:cnoremenu*
E330 *E327* *E331* *E336* *E333*
E328 *E329* *E337* *E792*
To create a new menu item, use the ":menu" commands. They are mostly like
the ":map" set of commands but the first argument is a menu item name, given
as a path of menus and submenus with a '.' between them, e.g.: >

```
:menu File.Save  :w<CR>
:inoremenu File.Save  <C-O>:w<CR>
:menu Edit.Big\ Changes.Delete\ All\ Spaces  :%s/[ ^I]//g<CR>
```

This last one will create a new item in the menu bar called "Edit", holding
the mouse button down on this will pop up a menu containing the item
"Big Changes", which is a sub-menu containing the item "Delete All Spaces",
which when selected, performs the operation.

Special characters in a menu name:

&	The next character is the shortcut key. Make sure each shortcut key is only used once in a (sub)menu. If you want to insert a literal "&" in the menu name use "&&".
<Tab>	Separates the menu name from right-aligned text. This can be used to show the equivalent typed command. The text "<Tab>" can be used here for convenience. If you are using a real tab, don't forget to put a backslash before it!

Example: >

```
   :amenu &File.&Open<Tab>:e  :browse e<CR>
```

[typed literally]
With the shortcut "F" (while keeping the <Alt> key pressed), and then "O",
this menu can be used. The second part is shown as "Open :e". The ":e"
is right aligned, and the "O" is underlined, to indicate it is the shortcut.

The ":amenu" command can be used to define menu entries for all modes at once.
To make the command work correctly, a character is automatically inserted for
some modes:

mode	inserted	appended	~
Normal	nothing	nothing	
Visual	<C-C>	<C-\><C-G>	
Insert	<C-\><C-O>		
Cmdline	<C-C>	<C-\><C-G>	
Op-pending	<C-C>	<C-\><C-G>	

Appending CTRL-\ CTRL-G is for going back to insert mode when 'insertmode' is
set. |CTRL-_CTRL-G|

Example: >

```
   :amenu File.Next     :next^M
```

is equal to: >

```
   :nmenu File.Next     :next^M
   :vmenu File.Next     ^C:next^M^\^G
   :imenu File.Next     ^\^O:next^M
   :cmenu File.Next     ^C:next^M^\^G
   :omenu File.Next     ^C:next^M^\^G
```

Careful: In Insert mode this only works for a SINGLE Normal mode command,
because of the CTRL-O. If you have two or more commands, you will need to use
the ":imenu" command. For inserting text in any mode, you can use the
expression register: >

```
   :amenu Insert.foobar   "='foobar'<CR>P
```

Note that the '<' and 'k' flags in 'cpoptions' also apply here (when
included they make the <> form and raw key codes not being recognized).

Note that <Esc> in Cmdline mode executes the command, like in a mapping. This
is Vi compatible. Use CTRL-C to quit Cmdline mode.

:menu-<silent> *:menu-silent*

To define a menu which will not be echoed on the command line, add
"<silent>" as the first argument. Example: >
	:menu <silent> Settings.Ignore\ case :set ic<CR>
The ":set ic" will not be echoed when using this menu. Messages from the
executed command are still given though. To shut them up too, add a ":silent"
in the executed command: >
	:menu <silent> Search.Header :exe ":silent normal /Header\r"<CR>
"<silent>" may also appear just after "<special>" or "<script>".

:menu-<special> *:menu-special*
Define a menu with <> notation for special keys, even though the "<" flag
may appear in 'cpoptions'. This is useful if the side effect of setting
'cpoptions' is not desired. Example: >
	:menu <special> Search.Header /Header<CR>
"<special>" must appear as the very first argument to the ":menu" command or
just after "<silent>" or "<script>".

:menu-<script> *:menu-script*
The "to" part of the menu will be inspected for mappings. If you don't want
this, use the ":noremenu" command (or the similar one for a specific mode).
If you do want to use script-local mappings, add "<script>" as the very first
argument to the ":menu" command or just after "<silent>" or "<special>".

menu-priority
You can give a priority to a menu. Menus with a higher priority go more to
the right. The priority is given as a number before the ":menu" command.
Example: >
	:80menu Buffer.next :bn<CR>

The default menus have these priorities:
	File		10
	Edit		20
	Tools		40
	Syntax		50
	Buffers		60
	Window		70
	Help		9999

When no or zero priority is given, 500 is used.
The priority for the PopUp menu is not used.

The Help menu will be placed on the far right side of the menu bar on systems
which support this (Motif and GTK+). For GTK+ 2 and 3, this is not done
anymore because right-aligning the Help menu is now discouraged UI design.

You can use a priority higher than 9999, to make it go after the Help menu,
but that is non-standard and is discouraged. The highest possible priority is
about 32000. The lowest is 1.

sub-menu-priority
The same mechanism can be used to position a sub-menu. The priority is then
given as a dot-separated list of priorities, before the menu name: >
	:menu 80.500 Buffer.next :bn<CR>
Giving the sub-menu priority is only needed when the item is not to be put
in a normal position. For example, to put a sub-menu before the other items: >
	:menu 80.100 Buffer.first :brew<CR>
Or to put a sub-menu after the other items, and further items with default
priority will be put before it: >
	:menu 80.900 Buffer.last :blast<CR>
When a number is missing, the default value 500 will be used: >
	:menu .900 myMenu.test :echo "text"<CR>
The menu priority is only used when creating a new menu. When it already

existed, e.g., in another mode, the priority will not change. Thus, the
priority only needs to be given the first time a menu is used.
An exception is the PopUp menu. There is a separate menu for each mode
(Normal, Op-pending, Visual, Insert, Cmdline). The order in each of these
menus can be different. This is different from menu-bar menus, which have
the same order for all modes.
NOTE: sub-menu priorities currently don't work for all versions of the GUI.

menu-separator *E332*

Menu items can be separated by a special item that inserts some space between
items. Depending on the system this is displayed as a line or a dotted line.
These items must start with a '-' and end in a '-'. The part in between is
used to give it a unique name. Priorities can be used as with normal items.
Example: >

```
:menu Example.item1     :do something
:menu Example.-Sep-     :
:menu Example.item2     :do something different
```

Note that the separator also requires a rhs. It doesn't matter what it is,
because the item will never be selected. Use a single colon to keep it
simple.

gui-toolbar

The toolbar is currently available in the Win32, Athena, Motif, GTK+ (X11),
and Photon GUI. It should turn up in other GUIs in due course. The
default toolbar is setup in menu.vim.
The display of the toolbar is controlled by the 'guioptions' letter 'T'. You
can thus have menu & toolbar together, or either on its own, or neither.
The appearance is controlled by the 'toolbar' option. You can choose between
an image, text or both.

toolbar-icon

The toolbar is defined as a special menu called ToolBar, which only has one
level. Vim interprets the items in this menu as follows:

1) If an "icon=" argument was specified, the file with this name is used.
   The file can either be specified with the full path or with the base name.
   In the last case it is searched for in the "bitmaps" directory in
   'runtimepath', like in point 3. Examples: >

```
:amenu icon=/usr/local/pixmaps/foo_icon.xpm ToolBar.Foo :echo "Foo"<CR>
:amenu icon=FooIcon ToolBar.Foo :echo "Foo"<CR>
```

<   Note that in the first case the extension is included, while in the second
   case it is omitted.
   If the file cannot be opened the next points are tried.
   A space in the file name must be escaped with a backslash.
   A menu priority must come _after_ the icon argument: >

```
:amenu icon=foo 1.42 ToolBar.Foo :echo "42!"<CR>
```

2) An item called 'BuiltIn##', where ## is a number, is taken as number ## of
   the built-in bitmaps available in Vim. Currently there are 31 numbered
   from 0 to 30 which cover most common editing operations |builtin-tools|. >

```
:amenu ToolBar.BuiltIn22 :call SearchNext("back")<CR>
```

3) An item with another name is first searched for in the directory
   "bitmaps" in 'runtimepath'. If found, the bitmap file is used as the
   toolbar button image. Note that the exact filename is OS-specific: For
   example, under Win32 the command >

```
:amenu ToolBar.Hello :echo "hello"<CR>
```

<   would find the file 'hello.bmp'. Under GTK+/X11 it is 'Hello.xpm'. With
   GTK+ 2 the files 'Hello.png', 'Hello.xpm' and 'Hello.bmp' are checked for
   existence, and the first one found would be used.
   For MS-Windows and GTK+ 2 the bitmap is scaled to fit the button. For
   MS-Windows a size of 18 by 18 pixels works best.
   For MS-Windows the bitmap should have 16 colors with the standard palette.
   The light grey pixels will be changed to the Window frame color and the
   dark grey pixels to the window shadow color. More colors might also work,

```
    depending on your system.
4)  If the bitmap is still not found, Vim checks for a match against its list
    of built-in names.  Each built-in button image has a name.
    So the command >
        :amenu ToolBar.Open :e
<   will show the built-in "open a file" button image if no open.bmp exists.
    All the built-in names can be seen used in menu.vim.
5)  If all else fails, a blank, but functioning, button is displayed.

                                                            *builtin-tools*
nr  Name                Normal action  ~
00  New                 open new window
01  Open                browse for file to open in current window
02  Save                write buffer to file
03  Undo                undo last change
04  Redo                redo last undone change
05  Cut                 delete selected text to clipboard
06  Copy                copy selected text to clipboard
07  Paste               paste text from clipboard
08  Print               print current buffer
09  Help                open a buffer on Vim's builtin help
10  Find                start a search command
11  SaveAll             write all modified buffers to file
12  SaveSesn            write session file for current situation
13  NewSesn             write new session file
14  LoadSesn            load session file
15  RunScript           browse for file to run as a Vim script
16  Replace             prompt for substitute command
17  WinClose            close current window
18  WinMax              make current window use many lines
19  WinMin              make current window use few lines
20  WinSplit            split current window
21  Shell               start a shell
22  FindPrev            search again, backward
23  FindNext            search again, forward
24  FindHelp            prompt for word to search help for
25  Make                run make and jump to first error
26  TagJump             jump to tag under the cursor
27  RunCtags            build tags for files in current directory
28  WinVSplit           split current window vertically
29  WinMaxWidth         make current window use many columns
30  WinMinWidth         make current window use few columns

                                        *hidden-menus* *win32-hidden-menus*
In the Win32 and GTK+ GUI, starting a menu name with ']' excludes that menu
from the main menu bar.  You must then use the |:popup| or |:tearoff| command
to display it.

                                        *window-toolbar* *WinBar*
Each window can have a local toolbar.  This uses the first line of the window,
thus reduces the space for the text by one line.  The items in the toolbar
must start with "WinBar".

Only text can be used.  When using Unicode, special characters can be used to
make the items look like icons.

If the items do not fit then the last ones cannot be used.  The toolbar does
not wrap.

Note that Vim may be in any mode when executing these commands.  The menu
should be defined for Normal mode and will be executed without changing the
current mode. Thus if the current window is in Visual mode and the menu
```

```
command does not intentionally change the mode, Vim will remain in Visual
mode.  Best is to use `:nnoremenu` to avoid side effects.

Example for debugger tools: >
        nnoremenu 1.10 WinBar.Step :Step<CR>
        nnoremenu 1.20 WinBar.Next :Next<CR>
        nnoremenu 1.30 WinBar.Finish :Finish<CR>
        nnoremenu 1.40 WinBar.Cont :Continue<CR>
<
The window toolbar uses the ToolbarLine and ToolbarButton highlight groups.

When splitting the window the window toolbar is not copied to the new window.

                                                        *popup-menu*
In the Win32, GTK+, Motif, Athena and Photon GUI, you can define the
special menu "PopUp".  This is the menu that is displayed when the right mouse
button is pressed, if 'mousemodel' is set to popup or popup_setpos.
Example: >
    nnoremenu 1.40 PopUp.&Paste "+gP
    menu PopUp


5.3 Showing What Menus Are Mapped To                    *showing-menus*

To see what an existing menu is mapped to, use just one argument after the
menu commands (just like you would with the ":map" commands).  If the menu
specified is a submenu, then all menus under that hierarchy will be shown.
If no argument is given after :menu at all, then ALL menu items are shown
for the appropriate mode (e.g., Command-line mode for :cmenu).

Special characters in the list, just before the rhs:
*       The menu was defined with "nore" to disallow remapping.
&       The menu was defined with "<script>" to allow remapping script-local
        mappings only.
-       The menu was disabled.

Note that hitting <Tab> while entering a menu name after a menu command may
be used to complete the name of the menu item.


5.4 Executing Menus                                     *execute-menus*

                                                *:em*  *:emenu* *E334* *E335*
:[range]em[enu] {menu}          Execute {menu} from the command line.
                                The default is to execute the Normal mode
                                menu.  If a range is specified, it executes
                                the Visual mode menu.
                                If used from <c-o>, it executes the
                                insert-mode menu Eg: >
        :emenu File.Exit

If the console-mode vim has been compiled with WANT_MENU defined, you can
use :emenu to access useful menu items you may have got used to from GUI
mode.  See 'wildmenu' for an option that works well with this.  See
|console-menus| for an example.

When using a range, if the lines match with '<,'>, then the menu is executed
using the last visual selection.


5.5 Deleting Menus                                      *delete-menus*
```

:unme *:unmenu*
:aun *:aunmenu*
:nunme *:nunmenu*
:ounme *:ounmenu*
:vunme *:vunmenu*
:xunme *:xunmenu*
:sunme *:sunmenu*
:iunme *:iunmenu*
:cunme *:cunmenu*

To delete a menu item or a whole submenu, use the unmenu commands, which are analogous to the unmap commands. Eg: >

```
:unmenu! Edit.Paste
```

This will remove the Paste item from the Edit menu for Insert and Command-line modes.

Note that hitting <Tab> while entering a menu name after an umenu command may be used to complete the name of the menu item for the appropriate mode.

To remove all menus use: *:unmenu-all* >

```
:unmenu *       " remove all menus in Normal and visual mode
:unmenu! *      " remove all menus in Insert and Command-line mode
:aunmenu *      " remove all menus in all modes
```

If you want to get rid of the menu bar: >

```
:set guioptions-=m
```

5.6 Disabling Menus *disable-menus*

:menu-disable *:menu-enable*

If you do not want to remove a menu, but disable it for a moment, this can be done by adding the "enable" or "disable" keyword to a ":menu" command. Examples: >

```
:menu disable &File.&Open\.\.\.
:amenu enable *
:amenu disable &Tools.*
```

The command applies to the modes as used with all menu commands. Note that characters like "&" need to be included for translated names to be found. When the argument is "*", all menus are affected. Otherwise the given menu name and all existing submenus below it are affected.

5.7 Examples for Menus *menu-examples*

Here is an example on how to add menu items with menu's! You can add a menu item for the keyword under the cursor. The register "z" is used. >

```
:nmenu Words.Add\ Var         wb"zye:menu! Words.<C-R>z <C-R>z<CR>
:nmenu Words.Remove\ Var      wb"zye:unmenu! Words.<C-R>z<CR>
:vmenu Words.Add\ Var         "zy:menu! Words.<C-R>z <C-R>z <CR>
:vmenu Words.Remove\ Var      "zy:unmenu! Words.<C-R>z<CR>
:imenu Words.Add\ Var         <Esc>wb"zye:menu! Words.<C-R>z <C-R>z<CR>a
:imenu Words.Remove\ Var      <Esc>wb"zye:unmenu! Words.<C-R>z<CR>a
```

(the rhs is in <> notation, you can copy/paste this text to try out the mappings, or put these lines in your gvimrc; "<C-R>" is CTRL-R, "<CR>" is the <CR> key. |<>|)

5.8 Tooltips & Menu tips

See section |42.4| in the user manual.

:tmenu *:tm*

:tm[enu] {menupath} {rhs} Define a tip for a menu or tool. {only in X11 and Win32 GUI}

:tm[enu] [menupath] List menu tips. {only in X11 and Win32 GUI}

:tunmenu *:tu*

:tu[nmenu] {menupath} Remove a tip for a menu or tool. {only in X11 and Win32 GUI}

When a tip is defined for a menu item, it appears in the command-line area when the mouse is over that item, much like a standard Windows menu hint in the status bar. (Except when Vim is in Command-line mode, when of course nothing is displayed.)
When a tip is defined for a ToolBar item, it appears as a tooltip when the mouse pauses over that button, in the usual fashion. Use the |hl-Tooltip| highlight group to change its colors.

A "tip" can be defined for each menu item. For example, when defining a menu item like this: >
:amenu MyMenu.Hello :echo "Hello"<CR>
The tip is defined like this: >
:tmenu MyMenu.Hello Displays a greeting.
And delete it with: >
:tunmenu MyMenu.Hello

Tooltips are currently only supported for the X11 and Win32 GUI. However, they should appear for the other gui platforms in the not too distant future.

The ":tmenu" command works just like other menu commands, it uses the same arguments. ":tunmenu" deletes an existing menu tip, in the same way as the other unmenu commands.

If a menu item becomes invalid (i.e. its actions in all modes are deleted) Vim deletes the menu tip (and the item) for you. This means that :aunmenu deletes a menu item - you don't need to do a :tunmenu as well.

5.9 Popup Menus

In the Win32 and GTK+ GUI, you can cause a menu to popup at the cursor. This behaves similarly to the PopUp menus except that any menu tree can be popped up.

This command is for backwards compatibility, using it is discouraged, because it behaves in a strange way.

:popup *:popu*

:popu[p] {name} Popup the menu {name}. The menu named must have at least one subentry, but need not appear on the menu-bar (see |hidden-menus|). {only available for Win32 and GTK GUI}

:popu[p]! {name} Like above, but use the position of the mouse pointer instead of the cursor.

Example: >
:popup File
will make the "File" menu (if there is one) appear at the text cursor (mouse

```
pointer if ! was used). >

        :amenu ]Toolbar.Make    :make<CR>
        :popup ]Toolbar
This creates a popup menu that doesn't exist on the main menu-bar.

Note that a menu that starts with ']' will not be displayed.

==============================================================================
6. Extras                                               *gui-extras*

This section describes other features which are related to the GUI.

- With the GUI, there is no wait for one second after hitting escape, because
  the key codes don't start with <Esc>.

- Typing ^V followed by a special key in the GUI will insert "<Key>", since
  the internal string used is meaningless.  Modifiers may also be held down to
  get "<Modifiers-Key>".

- In the GUI, the modifiers SHIFT, CTRL, and ALT (or META) may be used within
  mappings of special keys and mouse events.  E.g.: :map <M-LeftDrag> <LeftDrag>

- In the GUI, several normal keys may have modifiers in mappings etc, these
  are <Space>, <Tab>, <NL>, <CR>, <Esc>.

- To check in a Vim script if the GUI is being used, you can use something
  like this: >

        if has("gui_running")
           echo "yes, we have a GUI"
        else
           echo "Boring old console"
        endif
<                                                       *setting-guifont*
- When you use the same vimrc file on various systems, you can use something
  like this to set options specifically for each type of GUI: >

        if has("gui_running")
            if has("gui_gtk2")
                :set guifont=Luxi\ Mono\ 12
            elseif has("x11")
                " Also for GTK 1
                :set guifont=*-lucidatypewriter-medium-r-normal-*-*-180-*-*-m-*-*
            elseif has("gui_win32")
                :set guifont=Luxi_Mono:h12:cANSI
            endif
        endif

A recommended Japanese font is MS Mincho.  You can find info here:
http://www.lexikan.com/mincho.htm

==============================================================================
7. Shell Commands                                       *gui-shell*

For the X11 GUI the external commands are executed inside the gvim window.
See |gui-pty|.

WARNING: Executing an external command from the X11 GUI will not always
work.  "normal" commands like "ls", "grep" and "make" mostly work fine.
Commands that require an intelligent terminal like "less" and "ispell" won't
work.  Some may even hang and need to be killed from another terminal.  So be
```

careful!

For the Win32 GUI the external commands are executed in a separate window.
See |gui-shell-win32|.

 vim:tw=78:sw=4:ts=8:ft=help:norl:
gui_w32.txt   For Vim version 8.0. Last change: 2014 Dec 20

VIM REFERENCE MANUAL   by Bram Moolenaar

Vim's Win32 Graphical User Interface *gui-w32* *win32-gui*

Other relevant documentation:
|gui.txt| For generic items of the GUI.
|os_win32.txt| For Win32 specific items.

{Vi does not have a Windows GUI}

==
1. Starting the GUI *gui-w32-start*

The Win32 GUI version of Vim will always start the GUI, no matter how you
start it or what it's called.

The GUI will always run in the Windows subsystem. Mostly shells automatically
return with a command prompt after starting gvim. If not, you should use the
"start" command: >
	start gvim [options] file ..

Note: All fonts (bold, italic) must be of the same size!!! If you don't do
this, text will disappear or mess up the display. Vim does not check the font
sizes. It's the size in screen pixels that must be the same. Note that some
fonts that have the same point size don't have the same pixel size!
Additionally, the positioning of the fonts must be the same (ascent and
descent).

The Win32 GUI has an extra menu item: "Edit/Select Font". It brings up the
standard Windows font selector.

Setting the menu height doesn't work for the Win32 GUI.

gui-win32-maximized
If you want Vim to start with a maximized window, add this command to your
vimrc or gvimrc file: >
	au GUIEnter * simalt ~x
<

Using Vim as a plugin *gui-w32-windowid*

When gvim starts up normally, it creates its own top level window. If you
pass Vim the command-line option |--windowid| with a decimal or hexadecimal

value, Vim will create a window that is a child of the window with the given ID. This enables Vim to act as a plugin in another application. This really is a programmer's interface, and is of no use without a supporting application to spawn Vim correctly.

==
2. Vim as default editor *vim-default-editor*

To set Vim as the default editor for a file type:
1. Start a Windows Explorer
2. Choose View/Options -> File Types
3. Select the path to gvim for every file type that you want to use it for.
   (you can also use three spaces in the file type field, for files without an
   extension).
   In the "open" action, use: >
        gvim "%1"
< The quotes are required for using file names with embedded spaces.
   You can also use this: >
        gvim "%L"
< This should avoid short (8.3 character) file names in some situations. But
   I'm not sure if this works everywhere.

When you open a file in Vim by double clicking it, Vim changes to that file's directory.

If you want Vim to start full-screen, use this for the Open action: >
        gvim -c "simalt ~x" "%1"

Another method, which also works when you put Vim in another directory (e.g., when you have got a new version):
1. select a file you want to use Vim with
2. <Shift-F10>
3. select "Open With..." menu entry
4. click "Other..."
5. browse to the (new) location of Vim and click "Open"
6. make "Always Use this program..." checked
7. <OK>

send-to-menu *sendto*
You can also install Vim in the "Send To" menu:
1. Start a Windows Explorer
2. Navigate to your sendto directory:
   Windows NT: %windir%\profiles\%user%\sendto (e.g.
               "c:\winnt\profiles\mattha\sendto")
   Windows XP: C:\Documents and Settings\%user%\SendTo
   Windows Vista: C:\Users\%user%\AppData\Roaming\Microsoft\Windows\SendTo .
3. Right-click in the file pane and select New->Shortcut
4. Follow the shortcut wizard, using the full path to VIM/GVIM.

When you 'send a file to Vim', Vim changes to that file's directory. Note, however, that any long directory names will appear in their short (MS-DOS) form. This is a limitation of the Windows "Send To" mechanism.

notepad
You could replace notepad.exe with gvim.exe, but that has a few side effects. Some programs rely on notepad arguments, which are not recognized by Vim. For example "notepad -p" is used by some applications to print a file. It's better to leave notepad where it is and use another way to start Vim.

win32-popup-menu
A more drastic approach is to install an "Edit with Vim" entry in the popup menu for the right mouse button. With this you can edit any file with Vim.

This can co-exist with the file associations mentioned above. The difference is that the file associations will make starting Vim the default action. With the "Edit with Vim" menu entry you can keep the existing file association for double clicking on the file, and edit the file with Vim when you want. For example, you can associate "*.mak" with your make program. You can execute the makefile by double clicking it and use the "Edit with Vim" entry to edit the makefile.

You can select any files and right-click to see a menu option called "Edit with gvim". Choosing this menu option will invoke gvim with the file you have selected. If you select multiple files, you will find two gvim-related menu options:
"Edit with multiple gvims" -- one gvim for each file in the selection
"Edit with single gvim" -- one gvim for all the files in the selection
And if there already is a gvim running:
"Edit with existing gvim" -- edit the file with the running gvim

The "edit with existing Vim" entries can be disabled by adding an entry in the registry under HKLM\Software\Vim\Gvim, named DisableEditWithExisting, and with any value.

install-registry

You can add the "Edit with Vim" menu entry in an easy way by using the "install.exe" program. It will add several registry entries for you.

You can also do this by hand. This is complicated! Use the install.exe if you can.

1. Start the registry editor with "regedit".
2. Add these keys:

key	value name	value ~
HKEY_CLASSES_ROOT\CLSID\{51EEE242-AD87-11d3-9C1E-0090278BBD99}		
	{default}	Vim Shell Extension
HKEY_CLASSES_ROOT\CLSID\{51EEE242-AD87-11d3-9C1E-0090278BBD99}\InProcServer32		
	{default}	{path}\gvimext.dll
	ThreadingModel	Apartment
HKEY_CLASSES_ROOT*\shellex\ContextMenuHandlers\gvim		
	{default}	{51EEE242-AD87-11d3-9C1E-0090278BBD99}
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Shell Extensions\Approved		
	{51EEE242-AD87-11d3-9C1E-0090278BBD99}	Vim Shell Extension
HKEY_LOCAL_MACHINE\Software\Vim\Gvim		
	path	{path}\gvim.exe
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall\vim 5.6		
	DisplayName	Vim 5.6: Edit with Vim popup menu entry
	UninstallString	{path}\uninstal.exe

Replace {path} with the path that leads to the executable.
Don't type {default}, this is the value for the key itself.

To remove "Edit with Vim" from the popup menu, just remove the registry entries mentioned above. The "uninstal.exe" program can do this for you. You can also use the entry in the Windows standard "Add/Remove Programs" list.

If you notice that this entry overrules other file type associations, set those associations again by hand (using Windows Explorer, see above). This only seems to happen on some Windows NT versions (Windows bug?). Procedure:
1. Find the name of the file type. This can be done by starting the registry editor, and searching for the extension in \\HKEY_CLASSES_ROOT
2. In a Windows Explorer, use View/Options/File Types. Search for the file type in the list and click "Edit". In the actions list, you can select on

to be used as the default (normally the "open" action) and click on the
"Set Default" button.

Vim in the "Open With..." context menu *win32-open-with-menu*

If you use the Vim install program you have the choice to add Vim to the "Open
With..." menu. This means you can use Vim to edit many files. Not every file
(for unclear reasons...), thus the "Edit with Vim" menu entry is still useful.

One reason to add this is to be able to edit HTML files directly from Internet
Explorer. To enable this use the "Tools" menu, "Internet Options..." entry.
In the dialog select the "Programs" tab and select Vim in the "HTML editor"
choice. If it's not there than installing didn't work properly.

Doing this manually can be done with this script:

```
------------------------------------------------------------
REGEDIT4

[HKEY_CLASSES_ROOT\Applications\gvim.exe]

[HKEY_CLASSES_ROOT\Applications\gvim.exe\shell]

[HKEY_CLASSES_ROOT\Applications\gvim.exe\shell\edit]

[HKEY_CLASSES_ROOT\Applications\gvim.exe\shell\edit\command]
@="c:\\vim\\vim62\\gvim.exe \"%1\""

[HKEY_CLASSES_ROOT\.htm\OpenWithList\gvim.exe]

[HKEY_CLASSES_ROOT\*\OpenWithList\gvim.exe]

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

Change the "c:\\vim\\vim62" bit to where gvim.exe is actually located.

To uninstall this run the Vim uninstall program or manually delete the
registry entries with "regedit".

==
3. Using the clipboard *gui-clipboard*

Windows has a clipboard, where you can copy text to, and paste text from. Vim
supports this in several ways. For other systems see |gui-selections|.

The "* register reflects the contents of the clipboard. |quotestar|

When the "unnamed" string is included in the 'clipboard' option, the unnamed
register is the same. Thus you can yank to and paste from the clipboard
without prepending "* to commands.

The 'a' flag in 'guioptions' is not included by default. This means that text
is only put on the clipboard when an operation is performed on it. Just
Visually selecting text doesn't put it on the clipboard. When the 'a' flag is
included, the text is copied to the clipboard even when it is not operated
upon.

mswin.vim

To use the standard MS-Windows way of CTRL-X, CTRL-C and CTRL-V, use the
$VIMRUNTIME/mswin.vim script. You could add this line to your _vimrc file: >

```
source $VIMRUNTIME/mswin.vim
```

Since CTRL-C is used to copy the text to the clipboard, it can't be used to
cancel an operation. Use CTRL-Break for that.

CTRL-Z is used for undo. This means you can't suspend Vim with this key, use
|:suspend| instead (if it's supported at all).

CTRL-V-alternative *CTRL-Q*
Since CTRL-V is used to paste, you can't use it to start a blockwise Visual
selection. You can use CTRL-Q instead. You can also use CTRL-Q in Insert
mode and Command-line mode to get the old meaning of CTRL-V. But CTRL-Q
doesn't work for terminals when it's used for control flow.

NOTE: The clipboard support still has a number of bugs. See |todo|.

==
4. Shell Commands *gui-shell-win32*

Vim uses another window for external commands, to make it possible to run any
command. The external command gets its own environment for running, just like
it was started from a DOS prompt.

win32-vimrun
Executing an external command is done indirectly by the "vimrun" command. The
"vimrun.exe" must be in the path for this to work. Or it must be in the same
directory as the Vim executable. If "vimrun" cannot be found, the command is
executed directly, but then the DOS window closes immediately after the
external command has finished.
WARNING: If you close this window with the "X" button, and confirm the
question if you really want to kill the application, Vim may be killed too!
(This does not apply to commands run asynchronously with ":!start".)

The window in which the commands are executed will be the default you have set
up for "Console" in Control Panel.

win32-!start
Normally, Vim waits for a command to complete before continuing (this makes
sense for most shell commands which produce output for Vim to use). If you
want Vim to start a program and return immediately, you can use the following
syntax: >
        :!start [/min] {command}
The optional "/min" causes the window to be minimized.

==
5. Special colors *win32-colors*

On Win32, the normal DOS colors can be used. See |dos-colors|.

Additionally the system configured colors can also be used. These are known
by the names Sys_XXX, where XXX is the appropriate system color name, from the
following list (see the Win32 documentation for full descriptions). Case is
ignored.

Sys_3DDKShadow	Sys_3DFace	Sys_BTNFace
Sys_3DHilight	Sys_3DHighlight	Sys_BTNHilight
Sys_BTNHighlight	Sys_3DLight	Sys_3DShadow
Sys_BTNShadow	Sys_ActiveBorder	Sys_ActiveCaption
Sys_AppWorkspace	Sys_Background	Sys_Desktop
Sys_BTNText	Sys_CaptionText	Sys_GrayText
Sys_Highlight	Sys_HighlightText	Sys_InactiveBorder
Sys_InactiveCaption	Sys_InactiveCaptionText	Sys_InfoBK
Sys_InfoText	Sys_Menu	Sys_MenuText

```
Sys_ScrollBar           Sys_Window                      Sys_WindowFrame
Sys_WindowText

Probably the most useful values are
        Sys_Window          Normal window background
        Sys_WindowText      Normal window text
        Sys_Highlight       Highlighted background
        Sys_HighlightText   Highlighted text

These extra colors are also available:
Gray, Grey, LightYellow, SeaGreen, Orange, Purple, SlateBlue, Violet,

                                                                  *rgb.txt*
Additionally, colors defined by a "rgb.txt" file can be used.  This file is
well known from X11.  A few lines from it: >

  255 218 185             peach puff
  205 133  63             peru
  255 181 197             pink

This shows the layout of the file:  First the R, G and B value as a decimal
number, followed by the name of the color.  The four fields are separated by
spaces.

You can get an rgb.txt file from any X11 distribution.  It is located in a
directory like "/usr/X11R6/lib/X11/".  For Vim it must be located in the
$VIMRUNTIME directory.  Thus the file can be found with "$VIMRUNTIME/rgb.txt".

==============================================================================
                                            *gui-w32-dialogs* *dialog*
6. Windows dialogs & browsers

The Win32 GUI can use familiar Windows components for some operations, as well
as the traditional interface shared with the console version.


6.1 Dialogs

The dialogs displayed by the "confirm" family (i.e. the 'confirm' option,
|:confirm| command and |confirm()| function) are GUI-based rather than the
console-based ones used by other versions.  The 'c' flag in 'guioptions'
changes this.


6.2 File Browsers

When prepending ":browse" before file editing commands, a file requester is
used to allow you to select an existing file.  See |:browse|.


6.3 Tearoff Menus

The Win32 GUI emulates Motif's tear-off menus.  At the top of each menu you
will see a small graphic "rip here" sign.  Selecting it will cause a floating
window to be created with the same menu entries on it.  The floating menu can
then be accessed just as if it was the original (including sub-menus), but
without having to go to the menu bar each time.
This is most useful if you find yourself using a command buried in a sub-menu
over and over again.
The tearoff menus can be positioned where you like, and always stay just above
the Main Vim window.  You can get rid of them by closing them as usual; they
also of course close when you exit Vim.
```

:tearoff *:te*
:te[aroff] {name} Tear-off the menu {name}. The menu named must have at
                   least one subentry, but need not appear on the
                   menu-bar (see |win32-hidden-menus|).

Example: >
        :tearoff File
will make the "File" menu (if there is one) appear as a tearoff menu. >

        :amenu]Toolbar.Make   :make<CR>
        :tearoff]Toolbar
This creates a floating menu that doesn't exist on the main menu-bar.

Note that a menu that starts with ']' will not be displayed.

==
7. Command line arguments   *gui-w32-cmdargs*

Analysis of a command line into parameters is not standardised in MS Windows.
Gvim has to provide logic to analyse a command line. This logic is likely to
be different from the default logic provided by a compilation system used to
build vim. The differences relate to unusual double quote (") usage.
The arguments "C:\My Music\freude.txt" and "+/Sch\"iller" are handled in the
same way. The argument "+/Sch""iller" may be handled different by gvim and
vim, depending what it was compiled with.

The rules are:
      a) A parameter is a sequence of graphic characters.
      b) Parameters are separated by white space.
      c) A parameter can be enclosed in double quotes to include white space.
      d) A sequence of zero or more backslashes (\) and a double quote (")
        is special. The effective number of backslashes is halved, rounded
        down. An even number of backslashes reverses the acceptability of
        spaces and tabs, an odd number of backslashes produces a literal
        double quote.

So:
        "   is a special double quote
        \"   is a literal double quote
        \\"   is a literal backslash and a special double quote
        \\\"   is a literal backslash and a literal double quote
        \\\\"   is 2 literal backslashes and a special double quote
        \\\\\" is 2 literal backslashes and a literal double quote
        etc.

Example: >
        gvim "C:\My Music\freude" +"set ignorecase" +/"\"foo\\" +\"bar\\\"

opens "C:\My Music\freude" and executes the line mode commands: >
        set ignorecase; /"foo\ and /bar\"

==
8. Various   *gui-w32-various*

                                                         gui-w32-printing
The "File/Print" menu prints the text with syntax highlighting, see
|:hardcopy|. If you just want to print the raw text and have a default
printer installed this should also work: >
        :w >>prn

Vim supports a number of standard MS Windows features. Some of these are

```
detailed elsewhere: see |'mouse'|, |win32-hidden-menus|.

                                                        *drag-n-drop-win32*
You can drag and drop one or more files into the Vim window, where they will
be opened as normal.  See |drag-n-drop|.

                                                        *:simalt* *:sim*
:sim[alt] {key}         simulate pressing {key} while holding Alt pressed.
                        {not in Vi} {only for Win32 versions}
                        Note: ":si" means ":s" with the "i" flag.

Normally, Vim takes control of all Alt-<Key> combinations, to increase the
number of possible mappings.  This clashes with the standard use of Alt as the
key for accessing menus.
The quick way of getting standard behavior is to set the 'winaltkeys' option
to "yes".  This however prevents you from mapping Alt keys at all.
Another way is to set 'winaltkeys' to "menu".  Menu shortcut keys are then
handled by windows, other ALT keys can be mapped.  This doesn't allow a
dependency on the current state though.
To get round this, the :simalt command allows Vim (when 'winaltkeys' is not
"yes") to fake a Windows-style Alt keypress.  You can use this to map Alt key
combinations (or anything else for that matter) to produce standard Windows
actions.  Here are some examples: >

        :map <M-f> :simalt f<CR>
This makes Alt-F pop down the 'File' menu (with the stock Menu.vim) by
simulating the keystrokes Alt, F. >
        :map <M-Space> :simalt ~<CR>
This maps Alt-Space to pop down the system menu for the Vim window.  Note that
~ is used by simalt to represent the <Space> character. >
        :map <C-n> :simalt ~n<CR>
Maps Control-N to produce the keys Alt-Space followed by N.  This minimizes the
Vim window via the system menu.

Note that the key changes depending on the language you are using.

                                                *intellimouse-wheel-problems*
When using the Intellimouse mouse wheel causes Vim to stop accepting input, go
to:
        ControlPanel - Mouse - Wheel - UniversalScrolling - Exceptions

And add gvim to the list of applications.  This problem only appears to happen
with the Intellimouse driver 2.2 and when "Universal Scrolling" is turned on.


XPM support                                             *w32-xpm-support*

Gvim can be build on MS-Windows with support for XPM files.  |+xpm_w32|
See the Make_mvc.mak file for instructions, search for XPM.

To try out if XPM support works do this: >
        :help
        :exe 'sign define vimxpm icon=' . $VIMRUNTIME . '\\vim16x16.xpm'
        :exe 'sign place 1 line=1 name=vimxpm file=' . expand('%:p')
<

 vim:tw=78:sw=4:ts=8:ft=help:norl:
*gui_x11.txt*   For Vim version 8.0.  Last change: 2017 Jul 28


                  VIM REFERENCE MANUAL    by Bram Moolenaar
```

Vim's Graphical User Interface *gui-x11* *GUI-X11*
Athena *Motif*

Other relevant documentation:
|gui.txt| For generic items of the GUI.

{Vi does not have any of these commands}

==
1. Starting the X11 GUI *gui-x11-start* *E665*

Then you can run the GUI version of Vim in either of these ways:
gvim [options] [files...]
vim -g [options] [files...]

So if you call the executable "gvim", or make "gvim" a link to the executable, then the GUI version will automatically be used. Additional characters may be added after "gvim", for example "gvim-5".

You may also start up the GUI from within the terminal version by using one of these commands:
:gui [++opt] [+cmd] [-f|-b] [files...] *:gu* *:gui*
:gvim [++opt] [+cmd] [-f|-b] [files...] *:gv* *:gvim*
The "-f" option runs Vim in the foreground.
The "-b" option runs Vim in the background (this is the default).
Also see |++opt| and |+cmd|.

gui-fork
When the GUI is started, it does a fork() and exits the current process. When gvim was started from a shell this makes the shell accept further commands. If you don't want this (e.g. when using gvim for a mail program that waits for gvim to exit), start gvim with "gvim -f", "vim -gf" or use ":gui -f". Don't use "vim -fg", because "-fg" specifies the foreground color.

When using "gvim -f" and then ":gui", Vim will run in the foreground. The "-f" argument will be remembered. To force running Vim in the background use ":gui -b".

"gvim --nofork" does the same as "gvim -f".

E851 *E852*
When starting the GUI fails Vim will try to continue running in the terminal.

If you want the GUI to run in the foreground always, include the 'f' flag in 'guioptions'. |-f|.

==
2. GUI Resources *gui-resources* *.Xdefaults*

If using the Motif or Athena version of the GUI (not for the KDE, GTK+ or Win32 version), a number of X resources are available. You should use Vim's class "Vim" when setting these. They are as follows:

```
    Resource name       Meaning         ~

    reverseVideo        Boolean: should reverse video be used?
    background          Color of background.
    foreground          Color of normal text.
    scrollBackground    Color of trough portion of scrollbars.
    scrollForeground    Color of slider and arrow portions of scrollbars.
    menuBackground      Color of menu backgrounds.
    menuForeground      Color of menu foregrounds.
    tooltipForeground   Color of tooltip and balloon foreground.
    tooltipBackground   Color of tooltip and balloon background.

    font                Name of font used for normal text.
    boldFont            Name of font used for bold text.
    italicFont          Name of font used for italic text.
    boldItalicFont      Name of font used for bold, italic text.
    menuFont            Name of font used for the menus, used when compiled
                        without the |+xfontset| feature
    menuFontSet         Name of fontset used for the menus, used when compiled
                        with the |+xfontset| feature
    tooltipFont         Name of the font used for the tooltip and balloons.
                        When compiled with the |+xfontset| feature this is a
                        fontset name.

    geometry            Initial geometry to use for gvim's window (default
                        is same size as terminal that started it).
    scrollbarWidth      Thickness of scrollbars.
    borderWidth         Thickness of border around text area.
    menuHeight          Height of the menu bar (only for Athena).

A special font for italic, bold, and italic-bold text will only be used if
the user has specified one via a resource.  No attempt is made to guess what
fonts should be used for these based on the normal text font.

Note that the colors can also be set with the ":highlight" command, using the
"Normal", "Menu", "Tooltip", and "Scrollbar" groups.  Example: >
        :highlight Menu guibg=lightblue
        :highlight Tooltip guibg=yellow
        :highlight Scrollbar guibg=lightblue guifg=blue
        :highlight Normal guibg=grey90
<
                                                        *font-sizes*
Note: All fonts (except for the menu and tooltip) must be of the same size!!!
If you don't do this, text will disappear or mess up the display.  Vim does
not check the font sizes.  It's the size in screen pixels that must be the
same.  Note that some fonts that have the same point size don't have the same
pixel size!  Additionally, the positioning of the fonts must be the same
(ascent and descent).  You can check this with "xlsfonts -l {fontname}".

If any of these things are also set with Vim commands, e.g. with
":set guifont=Screen15", then this will override the X resources (currently
'guifont' is the only option that is supported).

Here is an example of what you might put in your ~/.Xdefaults file: >

        Vim*useSchemes:                 all
        Vim*sgiMode:                    true
        Vim*useEnhancedFSB:             true
        Vim.foreground:                 Black
        Vim.background:                 Wheat
        Vim*fontList:                   7x13
```

```
The first three of these are standard resources on Silicon Graphics machines
which make Motif applications look even better, highly recommended!

The "Vim*fontList" is to set the menu font for Motif.  Example: >
        Vim*menuBar*fontList:        -*-courier-medium-r-*-*-10-*-*-*-*-*-*-*
With Athena: >
        Vim*menuBar*SmeBSB*font:     -*-courier-medium-r-*-*-10-*-*-*-*-*-*-*
        Vim*menuBar*MenuButton*font: -*-courier-medium-r-*-*-10-*-*-*-*-*-*-*

NOTE: A more portable, and indeed more correct, way to specify the menu font
in either Motif or Athena is through the resource: >
        Vim.menuFont:        -*-courier-medium-r-*-*-10-*-*-*-*-*-*-*
Or, when compiled with the |+xfontset| feature: >
        Vim.menuFontSet:     -*-courier-medium-r-*-*-10-*-*-*-*-*-*-*

Don't use "Vim*geometry" in the defaults.  This will break the menus.  Use
"Vim.geometry" instead.

If you get an error message "Cannot allocate colormap entry for "gray60",
try adding this to your Vim resources (change the colors to your liking): >

        Vim*scrollBackground:           Black
        Vim*scrollForeground:           Blue

The resources can also be set with arguments to Vim:

    argument            meaning ~
                                                        *-gui*
   -display {display}   Run vim on {display}            *-display*
   -iconic              Start vim iconified             *-iconic*
   -background {color}  Use {color} for the background  *-background*
   -bg {color}          idem                            *-bg*
   -foreground {color}  Use {color} for normal text     *-foreground*
   -fg {color}          idem                            *-fg*
   -ul {color}          idem                            *-ul*
   -font {font}         Use {font} for normal text      *-font*
   -fn {font}           idem                            *-fn*
   -boldfont {font}     Use {font} for bold text        *-boldfont*
   -italicfont {font}   Use {font} for italic text      *-italicfont*
   -menufont {font}     Use {font} for menu items       *-menufont*
   -menufontset {fontset} Use {fontset} for menu items  *-menufontset*
   -mf {font}           idem                            *-mf*
   -geometry {geom}     Use {geom} for initial geometry *-geometry*
   -geom {geom}         idem, see |-geometry-example|   *-geom*
   -borderwidth {width} Use a border width of {width}   *-borderwidth*
   -bw {width}          idem                            *-bw*
                                                        *-scrollbarwidth*
   -scrollbarwidth {width}      Use a scrollbar width of {width}
   -sw {width}          idem                            *-sw*
   -menuheight {height} Use a menu bar height of {height} *-menuheight*
   -mh {height}         idem                            *-mh*
                        NOTE: On Motif the value is ignored, the menu height
                        is computed to fit the menus.
   -reverse             Use reverse video               *-reverse*
   -rv                  idem                            *-rv*
   +reverse             Don't use reverse video         *-+reverse*
   +rv                  idem                            *-+rv*
   -xrm {resource}      Set the specified resource      *-xrm*

Note about reverse video: Vim checks that the result is actually a light text
on a dark background.  The reason is that some X11 versions swap the colors,
```

and some don't. These two examples will both give yellow text on a blue
background:
    gvim -fg Yellow -bg Blue -reverse
    gvim -bg Yellow -fg Blue -reverse

-geometry-example
An example for the geometry argument: >
        gvim -geometry 80x63+8+100
This creates a window with 80 columns and 63 lines at position 8 pixels from
the left and 100 pixels from the top of the screen.

==
3. Shell Commands *gui-pty*

WARNING: Executing an external command from the GUI will not always work.
"normal" commands like "ls", "grep" and "make" mostly work fine. Commands
that require an intelligent terminal like "less" and "ispell" won't work.
Some may even hang and need to be killed from another terminal. So be
careful!

There are two ways to do the I/O with a shell command: Pipes and a pseudo-tty.
The default is to use a pseudo-tty. This should work best on most systems.

Unfortunately, the implementation of the pseudo-tty is different on every Unix
system. And some systems require root permission. To avoid running into
problems with a pseudo-tty when you least expect it, test it when not editing
a file. Be prepared to "kill" the started command or Vim. Commands like
":r !cat" may hang!

If using a pseudo-tty does not work for you, reset the 'guipty' option: >

        :set noguipty

Using a pipe should work on any Unix system, but there are disadvantages:
- Some shell commands will notice that a pipe is being used and behave
  differently. E.g., ":!ls" will list the files in one column.
- The ":sh" command won't show a prompt, although it will sort of work.
- When using ":make" it's not possible to interrupt with a CTRL-C.

Typeahead while the external command is running is often lost. This happens
both with a pipe and a pseudo-tty. This is a known problem, but it seems it
can't be fixed (or at least, it's very difficult).

gui-pty-erase
When your erase character is wrong for an external command, you should fix
this in your "~/.cshrc" file, or whatever file your shell uses for
initializations. For example, when you want to use backspace to delete
characters, but hitting backspaces produces "^H" instead, try adding this to
your "~/.cshrc": >
        stty erase ^H
The ^H is a real CTRL-H, type it as CTRL-V CTRL-H.

==
4. Various *gui-x11-various*

gui-x11-printing
The "File/Print" menu simply sends the current buffer to "lpr". No options or
whatever. If you want something else, you can define your own print command.
For example: >

  :10amenu File.Print :w !lpr -Php3
  :10vmenu File.Print :w !lpr -Php3

<
X11-icon
Vim uses a black&white icon by default when compiled with Motif or Athena. A colored Vim icon is included as $VIMRUNTIME/vim32x32.xpm. For GTK+, this is the builtin icon used. Unfortunately, how you should install it depends on your window manager. When you use this, remove the 'i' flag from 'guioptions', to remove the black&white icon: >
  :set guioptions-=i

If you use one of the fvwm* family of window managers simply add this line to your .fvwm2rc configuration file: >

  Style "vim"   Icon vim32x32.xpm

Make sure the icon file's location is consistent with the window manager's ImagePath statement. Either modify the ImagePath from within your .fvwm2rc or drop the icon into one the pre-defined directories: >

  ImagePath /usr/X11R6/include/X11/pixmaps:/usr/X11R6/include/X11/bitmaps

Note: older versions of fvwm use "IconPath" instead of "ImagePath".

For CDE "dtwm" (a derivative of Motif) add this line in the .Xdefaults: >
   Dtwm*Vim*iconImage: /usr/local/share/vim/vim32x32.xpm

For "mwm" (Motif window manager) the line would be: >
   Mwm*Vim*iconImage: /usr/local/share/vim/vim32x32.xpm

Mouse Pointers Available in X11 ~
X11_mouse_shapes
By using the |'mouseshape'| option, the mouse pointer can be automatically changed whenever Vim enters one of its various modes (e.g., Insert or Command). Currently, the available pointers are:

arrow	an arrow pointing northwest
beam	a I-like vertical bar
size	an arrow pointing up and down
busy	a wristwatch
blank	an invisible pointer
crosshair	a thin "+" sign
hand1	a dark hand pointing northeast
hand2	a light hand pointing northwest
pencil	a pencil pointing southeast
question	question_arrow
right_arrow	an arrow pointing northeast
up_arrow	an arrow pointing upwards

Additionally, any of the mouse pointers that are built into X11 may be used by specifying an integer from the X11/cursorfont.h include file.

If a name is used that exists on other systems, but not in X11, the default "arrow" pointer is used.

==
5. GTK version *gui-gtk* *GTK+* *GTK* *GTK3*

The GTK version of the GUI works a little bit different.

GTK does _not_ use the traditional X resource settings. Thus items in your ~/.Xdefaults or app-defaults files are not used.
Many of the traditional X command line arguments are not supported. (e.g.,

```
stuff like -bg, -fg, etc).  The ones that are supported are:

    command line argument   resource name       meaning ~
    -fn  or  -font          .font               font name for the text
    -geom  or  -geometry    .geometry           size of the gvim window
    -rv  or  -reverse       *reverseVideo       white text on black background
    -display                                    display to be used
    -fg -foreground {color}                     foreground color
    -bg -background {color}                     background color

To set the font, see |'guifont'|.  For GTK, there's also a menu option that
does this.

Additionally, there are these command line arguments, which are handled by GTK
internally.  Look in the GTK documentation for how they are used:
        --sync
        --gdk-debug
        --gdk-no-debug
        --no-xshm       (not in GTK+ 2)
        --xim-preedit   (not in GTK+ 2)
        --xim-status    (not in GTK+ 2)
        --gtk-debug
        --gtk-no-debug
        --g-fatal-warnings
        --gtk-module
        --display       (GTK+ counterpart of -display; works the same way.)
        --screen        (The screen number; for GTK+ 2.2 multihead support.)

These arguments are ignored when the |+netbeans_intg| feature is used:
        -xrm
        -mf

As for colors, Vim's color settings (for syntax highlighting) is still
done the traditional Vim way.  See |:highlight| for more help.

If you want to set the colors of remaining gui components (e.g., the
menubar, scrollbar, whatever), those are GTK specific settings and you
need to set those up in some sort of gtkrc file.  You'll have to refer
to the GTK documentation, however little there is, on how to do this.
See http://developer.gnome.org/doc/API/2.0/gtk/gtk-Resource-Files.html
for more information.


Tooltip Colors ~
                                                        *gtk-tooltip-colors*
Example, which sets the tooltip colors to black on light-yellow: >

        style "tooltips"
        {
                bg[NORMAL] = "#ffffcc"
                fg[NORMAL] = "#000000"
        }

        widget "gtk-tooltips*"          style "tooltips"

Write this in the file ~/.gtkrc and it will be used by GTK+.  For GTK+ 2
you might have to use the file ~/.gtkrc-2.0 instead, depending on your
distribution.

For GTK+ 3, an effect similar to the above can be obtained by adding the
following snippet of CSS code to $XDG_HOME_DIR/gtk-3.0/gtk.css (usually,
$HOME/.config/gtk-3.0/gtk.css):
```

```
For GTK+ 3 < 3.20: >

        .tooltip {
                background-color: #ffffcc;
                color: #000000;
        }
<
For GTK+ 3 >= 3.20: >

        tooltip {
            background-color: #ffffcc;
            text-shadow: none;
        }

        tooltip label {
            color: #2e3436;
        }
<

A Quick Look at GTK+ CSS ~
                                                              *gtk-css*
The contents of this subsection apply to GTK+ 3.20 or later which provides
stable support for GTK+ CSS:

        https://developer.gnome.org/gtk3/stable/theming.html

GTK+ uses CSS for styling and layout of widgets.  In this subsection, we'll
have a quick look at GTK+ CSS through simple, illustrative examples.

Example 1.  Empty Space Adjustment ~

By default, the toolbar and the tabline of the GTK+ 3 GUI are somewhat larger
than those of the GTK+ 2 GUI.  Some people may want to make them look similar
to the GTK+ 2 GUI in size.

To do that, we'll try reducing empty space around icons and labels that looks
apparently superfluous.

Add the following lines to $XDG_HOME_DIR/gtk-3.0/gtk.css (usually,
$HOME/.config/gtk-3.0/gtk.css): >

        toolbar button {
            margin-top: -2px;
            margin-right: 0px;
            margin-bottom: -2px;
            margin-left: 0px;

            padding-top: 0px;
            padding-right: 0px;
            padding-bottom: 0px;
            padding-left: 0px
        }

        notebook tab {
            margin-top: -1px;
            margin-right: 3px;
            margin-bottom: -1px;
            margin-left: 3px;

            padding-top: 0px;
            padding-right: 0px;
```

```
            padding-bottom: 0px;
            padding-left: 0px
        }
<
Since it's a CSS, they can be rewritten using shorthand: >

        toolbar button {
            margin: -2px 0px;
            padding: 0px;
        }

        notebook tab {
            margin: -1px 3px;
            padding: 0px
        }
<
Note: You might want to use 'toolbariconsize' to adjust the icon size, too.

Note: Depending on the icon theme and/or the font in use, some extra tweaks
may be needed for a satisfactory result.

Note: In addition to margin and padding, you can use border.  For details,
refer to the box model of CSS, e.g.,

        https://www.w3schools.com/css/css_boxmodel.asp

Example 2.  More Than Just Colors ~

GTK+ CSS supports gradients as well: >

        tooltip {
            background-image: -gtk-gradient(linear,
                                            0 0, 0 1,
                                            color-stop(0, #344752),
                                            color-stop(0.5, #546772),
                                            color-stop(1, #243742));
        }

        tooltip label {
            color: #f3f3f3;
        }
<
Gradients can be used to make a GUI element visually distinguishable from
others without relying on high contrast. Accordingly, effective use of them is
a useful technique to give a theme a sense of unity in color and luminance.

Note: Theming can be difficult since it must make every application look
equally good; making a single application more charming often gets others
unexpectedly less attractive or even deteriorates their usability.  Keep this
in mind always when you try improving a theme.


Using Vim as a GTK+ plugin ~
                                                        *gui-gtk-socketid*
When the GTK+ version of Vim starts up normally, it creates its own top level
window (technically, a 'GtkWindow').  GTK+ provides an embedding facility with
its GtkSocket and GtkPlug widgets.  If one GTK+ application creates a
GtkSocket widget in one of its windows, an entirely different GTK+ application
may embed itself into the first application by creating a top-level GtkPlug
widget using the socket's ID.

If you pass Vim the command-line option '--socketid' with a decimal or
```

hexadecimal value, Vim will create a GtkPlug widget using that value instead
of the normal GtkWindow. This enables Vim to act as a GTK+ plugin.

This really is a programmer's interface, and is of no use without a supporting
application to spawn the Vim correctly. For more details on GTK+ sockets, see
http://www.gtk.org/api/

Note that this feature requires the latest GTK version. GTK 1.2.10 still has
a small problem. The socket feature has not yet been tested with GTK+ 2 --
feel free to volunteer.

==
6. GNOME version *gui-gnome* *Gnome* *GNOME*

The GNOME GUI works just like the GTK+ version. See |GTK+| above for how it
works. It looks a bit different though, and implements one important feature
that's not available in the plain GTK+ GUI: Interaction with the session
manager. |gui-gnome-session|

These are the different looks:
- Uses GNOME dialogs (GNOME 1 only). The GNOME 2 GUI uses the same nice
  dialogs as the GTK+ 2 version.
- Uses the GNOME dock, so that the toolbar and menubar can be moved to
  different locations other than the top (e.g., the toolbar can be placed on
  the left, right, top, or bottom). The placement of the menubar and
  toolbar is only saved in the GNOME 2 version.
- That means the menubar and toolbar handles are back! Yeah! And the
  resizing grid still works too.

GNOME is compiled with if it was found by configure and the
--enable-gnome-check argument was used.

Note: Avoid use of --enable-gnome-check with GTK+ 3 GUI build. The
functionality mentioned above is consolidated in GTK+ 3.

GNOME session support ~
gui-gnome-session *gnome-session*
On logout, Vim shows the well-known exit confirmation dialog if any buffers
are modified. Clicking [Cancel] will stop the logout process. Otherwise the
current session is stored to disk by using the |:mksession| command, and
restored the next time you log in.

The GNOME session support should also work with the KDE session manager.
If you are experiencing any problems please report them as bugs.

Note: The automatic session save works entirely transparent, in order to
avoid conflicts with your own session files, scripts and autocommands. That
means in detail:
- The session file is stored to a separate directory (usually $HOME/.gnome2).
- 'sessionoptions' is ignored, and a hardcoded set of appropriate flags is
  used instead: >
        blank,curdir,folds,globals,help,options,tabpages,winsize
- The internal variable |v:this_session| is not changed when storing the
  session. Also, it is restored to its old value when logging in again.

The position and size of the GUI window is not saved by Vim since doing so
is the window manager's job. But if compiled with GTK+ 2 support, Vim helps
the WM to identify the window by restoring the window role (using the |--role|
command line argument).

==

7. KDE version *gui-kde* *kde* *KDE* *KVim*
gui-x11-kde

There is no KDE version of Vim. There has been some work on a port using the Qt toolkit, but it never worked properly and it has been abandoned. Work continues on Yzis: https://github.com/chrizel/Yzis.

==

8. Compiling *gui-x11-compiling*

If using X11, Vim's configure will by default first try to find the necessary GTK+ files on your system. When both GTK+ 2 and GTK+ 3 are available, GTK+ 2 will be chosen unless --enable-gui=gtk3 is passed explicitly to configure.

If the GTK+ files cannot be found, then the Motif files will be searched for. Finally, if this fails, the Athena files will be searched for. If all three fail, the GUI will be disabled.

For GTK+, Vim's configuration process uses pkg-config(1) to check if the GTK+ required for a specified build is properly installed and usable. Accordingly, it is a good idea to make sure before running configure that your system has a working pkg-config together with the .pc file of the required GTK+. For that, say, run the following on the command line to see if your pkg-config works with your GTK+ 2: >

```
    $ pkg-config --modversion gtk+-2.0
```

Replace gtk+-2.0 with gtk+-3.0 for GTK+ 3. If you get the correct version number of your GTK+, you can proceed; if not, you probably need to do some system administration chores to set up pkg-config and GTK+ correctly.

The GTK+ 2 GUI is built by default. Therefore, you usually don't need to pass any options such as --enable-gui=gtk2 to configure and build that.

Optionally, the GTK+ 2 GUI can consolidate the GNOME 2 support. This support is enabled by passing --enable-gnome-check to configure.

If you want to build the GTK+ 3 GUI, you have to pass --enable-gui=gtk3 explicitly to configure, and avoid passing --enable-gnome-check to that, as the functionality of the GNOME 2 support has already been consolidated in GTK+ 3.

Otherwise, if you are using Motif or Athena, when you have the Motif or Athena files in a directory where configure doesn't look, edit the Makefile to enter the names of the directories. Search for "GUI_INC_LOC" for an example to set the Motif directories, "CONF_OPT_X" for Athena.

gui-x11-gtk

Currently, Vim supports both GTK+ 2 and GTK+ 3.

The GTK+ 2 GUI requires GTK+ 2.2 or later.

Although the GTK+ 3 GUI is written in such a way that the source code can be compiled against all versions of the 3.x series, we recommend GTK+ 3.10 or later because of its substantial implementation changes in redraw done at that version.

gui-x11-motif

For Motif, you need at least Motif version 1.2 and/or X11R5. Motif 2.0 and X11R6 are OK. Motif 1.1 and X11R4 might work, no guarantee (there may be a few problems, but you might make it compile and run with a bit of work, please send me the patches if you do). The newest releases of LessTif have been reported to work fine too.

gui-x11-athena
The Athena version uses the Xaw widget set by default. If you have the 3D version, you might want to link with Xaw3d instead. This will make the menus look a bit better. Edit the Makefile and look for "XAW_LIB". The scrollbars will remain the same, because Vim has its own, which are already 3D (in fact, they look more like Motif).

gui-x11-neXtaw
The neXtaw version is mostly like Athena, but uses different widgets.

gui-x11-misc
In general, do not try to mix files from different GTK+, Motif, Athena and X11 versions. This will cause problems. For example, using header files for X11R5 with a library for X11R6 probably doesn't work (although the linking won't give an error message, Vim will crash later).

==
9. X11 selection mechanism *x11-selection*

If using X11, in either the GUI or an xterm with an X11-aware Vim, then Vim provides varied access to the X11 selection and clipboard. These are accessed by using the two selection registers "* and "+.

X11 provides two basic types of global store, selections and cut-buffers, which differ in one important aspect: selections are "owned" by an application, and disappear when that application (e.g., Vim) exits, thus losing the data, whereas cut-buffers, are stored within the X-server itself and remain until written over or the X-server exits (e.g., upon logging out).

The contents of selections are held by the originating application (e.g., upon a copy), and only passed on to another application when that other application asks for them (e.g., upon a paste).

The contents of cut-buffers are immediately written to, and are then accessible directly from the X-server, without contacting the originating application.

quoteplus *quote+*
There are three documented X selections: PRIMARY (which is expected to represent the current visual selection - as in Vim's Visual mode), SECONDARY (which is ill-defined) and CLIPBOARD (which is expected to be used for cut, copy and paste operations).

Of these three, Vim uses PRIMARY when reading and writing the "* register (hence when the X11 selections are available, Vim sets a default value for |'clipboard'| of "autoselect"), and CLIPBOARD when reading and writing the "+ register. Vim does not access the SECONDARY selection.

Examples: (assuming the default option values)
- Select a URL in Visual mode in Vim. Go to your browser and click the middle mouse button in the URL text field. The selected text will be inserted (hopefully!). Note: in Firefox you can set the middlemouse.contentLoadURL preference to true in about:config, then the selected URL will be used when pressing middle mouse button in most places in the window.
- Select some text in your browser by dragging with the mouse. Go to Vim and press the middle mouse button: The selected text is inserted.
- Select some text in Vim and do "+y. Go to your browser, select some text in a textfield by dragging with the mouse. Now use the right mouse button and select "Paste" from the popup menu. The selected text is overwritten by the text from Vim.

Note that the text in the "+ register remains available when making a Visual
selection, which makes other text available in the "* register. That allows
overwriting selected text.

x11-cut-buffer
There are, by default, 8 cut-buffers: CUT_BUFFER0 to CUT_BUFFER7. Vim only
uses CUT_BUFFER0, which is the one that xterm uses by default.

Whenever Vim is about to become unavailable (either via exiting or becoming
suspended), and thus unable to respond to another application's selection
request, it writes the contents of any owned selection to CUT_BUFFER0. If the
"+ CLIPBOARD selection is owned by Vim, then this is written in preference,
otherwise if the "* PRIMARY selection is owned by Vim, then that is written.

Similarly, when Vim tries to paste from "* or "+ (either explicitly, or, in
the case of the "* register, when the middle mouse button is clicked), if the
requested X selection is empty or unavailable, Vim reverts to reading the
current value of the CUT_BUFFER0.

Note that when text is copied to CUT_BUFFER0 in this way, the type of
selection (character, line or block) is always lost, even if it is a Vim which
later pastes it.

Xterm, by default, always writes visible selections to both PRIMARY and
CUT_BUFFER0. When it pastes, it uses PRIMARY if this is available, or else
falls back upon CUT_BUFFER0. For this reason, when cutting and pasting
between Vim and an xterm, you should use the "* register. Xterm doesn't use
CLIPBOARD, thus the "+ doesn't work with xterm.

Most newer applications will provide their current selection via PRIMARY ("*)
and use CLIPBOARD ("+) for cut/copy/paste operations. You thus have access to
both by choosing to use either of the "* or "+ registers.

vim:tw=78:sw=4:ts=8:ft=help:norl: