

Nirvana Editor (NEdit) Help Documentation

Table of Contents

<u>Nirvana Editor (NEdit) Help Documentation</u>	1
<u>Table of Contents</u>	1
<u>Getting Started</u>	1
<u>Editing an Existing File</u>	2
<u>Creating a New File</u>	2
<u>Backup Files</u>	2
<u>Shortcuts</u>	2
<u>Basic Operation</u>	4
<u>Selecting Text</u>	4
<u>Finding and Replacing Text</u>	5
<u>Searching Backwards</u>	5
<u>Selective Replacement</u>	5
<u>Cut and Paste</u>	5
<u>Using the Mouse</u>	6
<u>Button and Modifier Key Summary</u>	6
<u>Left Mouse Button</u>	7
<u>Right Mouse Button</u>	7
<u>Middle Mouse Button</u>	7
<u>Keyboard Shortcuts</u>	9
<u>Menu Accelerators</u>	9
<u>Menu Mnemonics</u>	9
<u>Keyboard Shortcuts within Dialogs</u>	9
<u>Labeled Function Keys</u>	10
<u>Modifier Keys (in general)</u>	10
<u>All Keyboards</u>	10
<u>PC Standard Keyboard</u>	11
<u>Specialty Keyboards</u>	12
<u>Shifting and Filling</u>	12
<u>Shift Left, Shift Right</u>	12
<u>Filling</u>	12
<u>File Format</u>	13
<u>Features for Programming</u>	15
<u>Programming with NEdit</u>	15
<u>Language Modes</u>	15
<u>Backlighting [EXPERIMENTAL]</u>	15
<u>Line Numbers</u>	15
<u>Matching Parentheses</u>	16
<u>Opening Included Files</u>	16
<u>Interface to Programming Tools</u>	16
<u>Tabs/Emulated Tabs</u>	16
<u>Changing the Tab Distance</u>	16
<u>Emulated Tabs</u>	17
<u>Auto/Smart Indent</u>	17
<u>Smart Indent</u>	17
<u>Auto-Indent</u>	18
<u>Block Indentation Adjustment</u>	18

Table of Contents

Features for Programming

<u>Syntax Highlighting</u>	18
<u>Finding Declarations (ctags)</u>	19
<u>Calltips</u>	19

Regular Expressions.....21

<u>Basic Regular Expression Syntax</u>	21
<u>The 'Any' Character</u>	21
<u>Character Classes</u>	21
<u> Anchors</u>	22
<u>Quantifiers</u>	22
<u>Alternation</u>	23
<u>Comments</u>	23
<u>Metacharacters</u>	23
<u>Escaping Metacharacters</u>	23
<u>Special Control Characters</u>	23
<u>Octal and Hex Escape Sequences</u>	24
<u>Shortcut Escape Sequences</u>	24
<u>Word Delimiter Tokens</u>	24
<u>Parenthetical Constructs</u>	24
<u>Capturing Parentheses</u>	25
<u>Non-Capturing Parentheses</u>	25
<u>Positive Look-Ahead</u>	25
<u>Negative Look-Ahead</u>	25
<u>Positive Look-Behind</u>	25
<u>Negative Look-Behind</u>	26
<u>Case Sensitivity</u>	26
<u>Matching Newlines</u>	26
<u>Notes on New Parenthetical Constructs</u>	27
<u>Back References</u>	27
<u>Substitution</u>	27
<u>Advanced Topics</u>	27
<u>Substitutions</u>	27
<u>Ambiguity</u>	28
<u>References</u>	28
<u>Example Regular Expressions</u>	28

Macro/Shell Extensions.....30

<u>Shell Commands and Filters</u>	30
<u>Learn/Replay</u>	30
<u>Repeating Actions and Learn/Replay Sequences</u>	30
<u>Macro Language</u>	31
<u>Syntax</u>	31
<u>Data Types</u>	32
<u>Variables</u>	32
<u>Functions and Subroutines</u>	33
<u>Operators and Expressions</u>	34
<u>Looping and Conditionals</u>	37

Table of Contents

Macro/Shell Extensions

<u>Macro Subroutines</u>	37
<u>Built in Variables</u>	38
<u>Built-in Subroutines</u>	41
<u>Deprecated Functions</u>	45
<u>Range Sets</u>	45
<u>Range set read-only variables</u>	46
<u>Range set functions</u>	46
<u>Highlighting Information</u>	47
<u>Action Routines</u>	49
<u>Actions Representing Menu Commands</u>	49
<u>Menu Action Routine Arguments</u>	50
<u>Window Preferences Actions</u>	51
<u>Keyboard-Only Actions</u>	53

Customizing.....60

<u>Customizing NEdit</u>	60
<u>Preferences</u>	60
<u>Preferences Menu</u>	60
<u>Preferences -> Default Settings Menu</u>	62
<u>Changing Font(s)</u>	64
<u>Customizing Menus</u>	64
<u>The NEdit Preferences File</u>	66
<u>Sharing Customizations with Other NEdit Users</u>	66
<u>X Resources</u>	67
<u>Selected X Resource Names</u>	67
<u>Key Binding</u>	73
<u>Key Binding in General</u>	73
<u>Key Binding Via Translations</u>	73
<u>Changing Menu Accelerator Keys</u>	74
<u>Highlighting Patterns</u>	74
<u>Writing Syntax Highlighting Patterns</u>	74
<u>Smart Indent Macros</u>	77
<u>NEdit Command Line</u>	77
<u>Client/Server Mode</u>	80
<u>Crash Recovery</u>	83
<u>Version</u>	83
<u>Distribution Policy</u>	84
<u>Mailing Lists</u>	88
<u>Problems/Defects</u>	88
<u>Solutions to Common Problems</u>	88
<u>Known Defects</u>	90
<u>Reporting Defects</u>	90

Nirvana Editor (NEdit) Help Documentation

Table of Contents

Getting Started

Basic Operation

Selecting Text
Finding and Replacing Text
Cut and Paste
Using the Mouse
Keyboard Shortcuts
Shifting and Filling
File Format

Features for Programming

Programming with NEdit
Tabs/Emulated Tabs
Auto/Smart Indent
Syntax Highlighting
Finding Declarations (ctags)
Calltips

Regular Expressions

Basic Regular Expression Syntax
Metacharacters
Parenthetical Constructs
Advanced Topics
Example Regular Expressions

Macro/Shell Extensions

Shell Commands and Filters
Learn/Replay
Macro Language
Macro Subroutines
Highlighting Information
Range Sets
Action Routines

Customizing

Customizing NEdit
Preferences
X Resources
Key Binding
Highlighting Patterns
Smart Indent Macros

NEdit Command Line

Client/Server Mode
Crash Recovery
Version
Distribution Policy
Mailing Lists
Problems/Defects

Getting Started

Welcome to NEdit!

NEdit is a standard GUI (Graphical User Interface) style text editor for programs and plain-text files. Users of Macintosh and MS Windows based text editors should find NEdit a familiar and comfortable environment. NEdit provides all of the standard menu, dialog, editing, and mouse support, as well as all of the standard shortcuts to which the users of modern GUI based environments are accustomed. For users of older style Unix editors, welcome to the world of mouse-based editing!

Help sections of interest to new users are listed under the "Basic Operation" heading in the top-level Help menu:

Selecting Text
Finding and Replacing Text
Cut and Paste
Using the Mouse
Keyboard Shortcuts
Shifting and Filling

Programmers should also read the introductory section under the "Features for Programming" section:

Programming with NEdit

If you get into trouble, the Undo command in the Edit menu can reverse any modifications that you make. NEdit does not change the file you are editing until you tell it to Save.

Editing an Existing File

To open an existing file, choose Open... from the file menu. Select the file that you want to open in the pop-up dialog that appears and click on OK. You may open any number of files at the same time. Each file will appear in its own editor window. Using Open... rather than re-typing the NEdit command and running additional copies of NEdit, will give you quick access to all of the files you have open via the Windows menu, and ensure that you don't accidentally open the same file twice. NEdit has no "main" window. It remains running as long as at least one editor window is open.

Creating a New File

If you already have an empty (Untitled) window displayed, just begin typing in the window. To create a new Untitled window, choose New from the File menu. To give the file a name and save its contents to the disk, choose Save or Save As... from the File menu.

Backup Files

NEdit maintains periodic backups of the file you are editing so that you can recover the file in the event of a problem such as a system crash, network failure, or X server crash. These files are saved under the name `~filename` (on Unix) or `_filename` (on VMS), where filename is the name of the file you were editing. If an NEdit process is killed, some of these backup files may remain in your directory. (To remove one of these files on Unix, you may have to prefix the `~` (tilde) character with a (backslash) to prevent the shell from interpreting it as a special character.)

Shortcuts

As you become more familiar with NEdit, substitute the control and function keys shown on the right side of the menus for pulling down menus with the mouse.

Dialogs are also streamlined so you can enter information quickly and without using the mouse*. To move the keyboard focus around a dialog, use the tab and arrow keys. One of the buttons in a dialog is usually drawn with a thick, indented, outline. This button can be activated by pressing Return or Enter. The Cancel or Dismiss button can be activated by pressing escape. For example, to replace the string "thing" with "things" type:

```
<ctrl-r>thing<tab>things<return>
```

To open a file named "whole_earth.c", type:

```
<ctrl-o>who<return>
```

(how much of the filename you need to type depends on the other files in the directory). See the section called "Keyboard Shortcuts" for more details.

Nirvana Editor (NEdit) Help Documentation

* Users who have set their keyboard focus mode to "pointer" should set "Popups Under Pointer" in the Default Settings menu to avoid the additional step of moving the mouse into the dialog.

Basic Operation

Selecting Text

NEdit has two general types of selections, primary (highlighted text), and secondary (underlined text). Selections can cover either a simple range of text between two points in the file, or they can cover a rectangular area of the file. Rectangular selections are only useful with non-proportional (fixed spacing) fonts.

To select text for copying, deleting, or replacing, press the left mouse button with the pointer at one end of the text you want to select, and drag it to the other end. The text will become highlighted. To select a whole word, double click (click twice quickly in succession). Double clicking and then dragging the mouse will select a number of words. Similarly, you can select a whole line or a number of lines by triple clicking or triple clicking and dragging. Quadruple clicking selects the whole file. After releasing the mouse button, you can still adjust a selection by holding down the shift key and dragging on either end of the selection. To delete the selected text, press delete or backspace. To replace it, begin typing.

To select a rectangle or column of text, hold the Ctrl key while dragging the mouse. Rectangular selections can be used in any context that normal selections can be used, including cutting and pasting, filling, shifting, dragging, and searching. Operations on rectangular selections automatically fill in tabs and spaces to maintain alignment of text within and to the right of the selection. Note that the interpretation of rectangular selections by Fill Paragraph is slightly different from that of other commands, the section titled "[Shifting and Filling](#)" has details.

The middle mouse button can be used to make an additional selection (called the secondary selection). As soon as the button is released, the contents of this selection will be copied to the insert position of the window where the mouse was last clicked (the destination window). This position is marked by a caret shaped cursor when the mouse is outside of the destination window. If there is a (primary) selection, adjacent to the cursor in the window, the new text will replace the selected text. Holding the shift key while making the secondary selection will move the text, deleting it at the site of the secondary selection, rather than copying it.

Selected text can also be dragged to a new location in the file using the middle mouse button. Holding the shift key while dragging the text will copy the selected text, leaving the original text in place. Holding the control key will drag the text in overlay mode.

Normally, dragging moves text by removing it from the selected position at the start of the drag, and inserting it at a new position relative to the mouse. Dragging a block of text over existing characters, displaces the characters to the end of the selection. In overlay mode, characters which are occluded by blocks of text being dragged are simply removed. When dragging non-rectangular selections, overlay mode also converts the selection to rectangular form, allowing it to be dragged outside of the bounds of the existing text.

The section "[Using the Mouse](#)" summarizes the mouse commands for making primary and secondary selections. Primary selections can also be made via keyboard commands, see "[Keyboard Shortcuts](#)".

Finding and Replacing Text

The Search menu contains a number of commands for finding and replacing text.

The Find... and Replace... commands present dialogs for entering text for searching and replacing. These dialogs also allow you to choose whether you want the search to be sensitive to upper and lower case, or whether to use the standard Unix pattern matching characters (regular expressions). Searches begin at the current text insertion position.

Find Again and Replace Again repeat the last find or replace command without prompting for search strings. To selectively replace text, use the two commands in combination: Find Again, then Replace Again if the highlighted string should be replaced, or Find Again again to go to the next string.

Find Selection searches for the text contained in the current primary selection (see [Selecting Text](#)). The selected text does not have to be in the current editor window, it may even be in another program. For example, if the word dog appears somewhere in a window on your screen, and you want to find it in the file you are editing, select the word dog by dragging the mouse across it, switch to your NEdit window and choose Find Selection from the Search menu.

Find Incremental is yet another variation on searching, where every character typed triggers a new search. Incremental searching is generally the quickest way to find something in a file, because it gives you the immediate feedback of seeing how your search is progressing, so you never need to type more than the minimally sufficient search string to reach your target.

Searching Backwards

Holding down the shift key while choosing any of the search or replace commands from the menu (or using the keyboard shortcut), will search in the reverse direction. Users who have set the search direction using the buttons in the search dialog, may find it a bit confusing that Find Again and Replace Again don't continue in the same direction as the original search (for experienced users, consistency of the direction implied by the shift key is more important).

Selective Replacement

To replace only some occurrences of a string within a file, choose Replace... from the Search menu, enter the string to search for and the string to substitute, and finish by pressing the Find button. When the first occurrence is highlighted, use either Replace Again (^T) to replace it, or Find Again (^G) to move to the next occurrence without replacing it, and continue in such a manner through all occurrences of interest.

To replace all occurrences of a string within some range of text, select the range (see [Selecting Text](#)), choose Replace... from the Search menu, type the string to search for and the string to substitute, and press the "R. in Selection" button in the dialog. Note that selecting text in the Replace... dialog will unselect the text in the window.

Cut and Paste

The easiest way to copy and move text around in your file or between windows, is to use the clipboard, an imaginary area that temporarily stores text and data. The Cut command removes the selected text (see

Selecting Text) from your file and places it in the clipboard. Once text is in the clipboard, the Paste command will copy it to the insert position in the current window. For example, to move some text from one place to another, select it by dragging the mouse over it, choose Cut to remove it, click the pointer to move the insert point where you want the text inserted, then choose Paste to insert it. Copy copies text to the clipboard without deleting it from your file. You can also use the clipboard to transfer text to and from other Motif programs and X programs which make proper use of the clipboard.

There are many other methods for copying and moving text within NEdit windows and between NEdit and other programs. The most common such method is clicking the middle mouse button to copy the primary selection (to the clicked position). Copying the selection by clicking the middle mouse button in many cases is the only way to transfer data to and from many X programs. Holding the Shift key while clicking the middle mouse button moves the text, deleting it from its original position, rather than copying it. Other methods for transferring text include secondary selections, primary selection dragging, keyboard-based selection copying, and drag and drop. These are described in detail in the sections: "Selecting Text", "Using the Mouse", and "Keyboard Shortcuts".

Using the Mouse

Mouse-based editing is what NEdit is all about, and learning to use the more advanced features like secondary selections and primary selection dragging will be well worth your while.

If you don't have time to learn everything, you can get by adequately with just the left mouse button: Clicking the left button moves the cursor. Dragging with the left button makes a selection. Holding the shift key while clicking extends the existing selection, or begins a selection between the cursor and the mouse. Double or triple clicking selects a whole word or a whole line.

This section will make more sense if you also read the section called, "Selecting Text", which explains the terminology of selections, that is, what is meant by primary, secondary, rectangular, etc.

Button and Modifier Key Summary

General meaning of mouse buttons and modifier keys:

Buttons

Button 1 (left)	Cursor position and primary selection
Button 2 (middle)	Secondary selections, and dragging and copying the primary selection
Button 3 (right)	Quick-access programmable menu and pan scrolling

Modifier keys

Shift	On primary selections, (left mouse button): Extends selection to the mouse pointer On secondary and copy operations, (middle): Toggles between move and copy
Ctrl	Makes selection rectangular or insertion

Nirvana Editor (NEdit) Help Documentation

columnar

Alt* (on release) Exchange primary and secondary selections.

Left Mouse Button

The left mouse button is used to position the cursor and to make primary selections.

Click	Moves the cursor
Double Click	Selects a whole word
Triple Click	Selects a whole line
Quad Click	Selects the whole file
Shift Click	Adjusts (extends or shrinks) the selection, or if there is no existing selection, begins a new selection between the cursor and the mouse.
Ctrl+Shift+Click	Adjusts (extends or shrinks) the selection rectangularly.
Drag	Selects text between where the mouse was pressed and where it was released.
Ctrl+Drag	Selects rectangle between where the mouse was pressed and where it was released.

Right Mouse Button

The right mouse button posts a programmable menu for frequently used commands.

Click/Drag	Pops up the background menu (programmed from Preferences -> Default Settings -> Customize Menus -> Window Background).
Ctrl+Drag	Pan scrolling. Scrolls the window both vertically and horizontally, as if you had grabbed it with your mouse.

Middle Mouse Button

The middle mouse button is for making secondary selections, and copying and dragging the primary selection.

Click	Copies the primary selection to the clicked position.
Shift+Click	Moves the primary selection to the clicked position, deleting it from its original position.

Nirvana Editor (NEdit) Help Documentation

Drag	1) Outside of the primary selection: Begins a secondary selection. 2) Inside of the primary selection: Moves the selection by dragging.
Ctrl+Drag	1) Outside of the primary selection: Begins a rectangular secondary selection. 2) Inside of the primary selection: Drags the selection in overlay mode (see below).

When the mouse button is released after creating a secondary selection:

No Modifiers	If there is a primary selection, replaces it with the secondary selection. Otherwise, inserts the secondary selection at the cursor position.
Shift	Move the secondary selection, deleting it from its original position. If there is a primary selection, the move will replace the primary selection with the secondary selection. Otherwise, moves the secondary selection to to the cursor position.
Alt*	Exchange the primary and secondary selections.

While moving the primary selection by dragging with the middle mouse button:

Shift	Leaves a copy of the original selection in place rather than removing it or blanking the area.
Ctrl	Changes from insert mode to overlay mode (see below).
Escape	Cancel drag in progress.

Overlay Mode: Normally, dragging moves text by removing it from the selected position at the start of the drag, and inserting it at a new position relative to to the mouse. When you drag a block of text over existing characters, the existing characters are displaced to the end of the selection. In overlay mode, characters which are occluded by blocks of text being dragged are simply removed. When dragging non-rectangular selections, overlay mode also converts the selection to rectangular form, allowing it to be dragged outside of the bounds of the existing text.

Mouse buttons 4 and 5 are usually represented by a mouse wheel nowadays. They are used to scroll up or down in the text window.

* The Alt key may be labeled Meta or Compose-Character on some keyboards. Some window managers, including default configurations of mwm, bind combinations of the Alt key and mouse buttons to window manager operations. In NEdit, Alt is only used on button release, so regardless of the window manager bindings for Alt-modified mouse buttons, you can still do the corresponding NEdit operation by using the Alt

key AFTER the initial mouse press, so that Alt is held while you release the mouse button. If you find this difficult or annoying, you can re-configure most window managers to skip this binding, or you can re-configure NEdit to use a different key combination.

Keyboard Shortcuts

Most of the keyboard shortcuts in NEdit are shown on the right hand sides of the pull-down menus. However, there are more which are not as obvious. These include; dialog button shortcuts; menu and dialog mnemonics; labeled keyboard keys, such as the arrows, page-up, page-down, and home; and optional Shift modifiers on accelerator keys, like [Shift]Ctrl+F.

Menu Accelerators

Pressing the key combinations shown on the right of the menu items is a shortcut for selecting the menu item with the mouse. Some items have the shift key enclosed in brackets, such as [Shift]Ctrl+F. This indicates that the shift key is optional. In search commands, including the shift key reverses the direction of the search. In Shift commands, it makes the command shift the selected text by a whole tab stop rather than by single characters.

Menu Mnemonics

Pressing the Alt key in combination with one of the underlined characters in the menu bar pulls down that menu. Once the menu is pulled down, typing the underlined characters in a menu item (without the Alt key) activates that item. With a menu pulled down, you can also use the arrow keys to select menu items, and the Space or Enter keys to activate them.

Keyboard Shortcuts within Dialogs

One button in a dialog is usually marked with a thick indented outline. Pressing the Return or Enter key activates this button.

All dialogs have either a Cancel or Dismiss button. This button can be activated by pressing the Escape (or Esc) key.

Pressing the tab key moves the keyboard focus to the next item in a dialog. Within an associated group of buttons, the arrow keys move the focus among the buttons. Shift+Tab moves backward through the items.

Most items in dialogs have an underline under one character in their name. Pressing the Alt key along with this character, activates a button as if you had pressed it with the mouse, or moves the keyboard focus to the associated text field or list.

You can select items from a list by using the arrow keys to move the selection and space to select.

In file selection dialogs, you can type the beginning characters of the file name or directory in the list to select files

Labeled Function Keys

The labeled function keys on standard workstation and PC keyboards, like the arrows, and page-up and page-down, are active in NEdit, though not shown in the pull-down menus.

Holding down the control key while pressing a named key extends the scope of the action that it performs. For example, Home normally moves the insert cursor the beginning of a line. Ctrl+Home moves it to the beginning of the file. Backspace deletes one character, Ctrl+Backspace deletes one word.

Holding down the shift key while pressing a named key begins or extends a selection. Combining the shift and control keys combines their actions. For example, to select a word without using the mouse, position the cursor at the beginning of the word and press Ctrl+Shift+RightArrow. The Alt key modifies selection commands to make the selection rectangular.

Under X and Motif, there are several levels of translation between keyboard keys and the actions they perform in a program. The "[Customizing NEdit](#)", and "[X Resources](#)" sections of the Help menu have more information on this subject. Because of all of this configurability, and since keyboards and standards for the meaning of some keys vary from machine to machine, the mappings may be changed from the defaults listed below.

Modifier Keys (in general)

Ctrl	Extends the scope of the action that the key would otherwise perform. For example, Home normally moves the insert cursor to the beginning of a line. Ctrl+Home moves it to the beginning of the file. Backspace deletes one character, Ctrl+Backspace deletes one word.
Shift	Extends the selection to the cursor position. If there's no selection, begins one between the old and new cursor positions.
Alt	When modifying a selection, makes the selection rectangular.

(For the effects of modifier keys on mouse button presses, see the section titled "[Using the Mouse](#)")

All Keyboards

Escape	Cancels operation in progress: menu selection, drag, selection, etc. Also equivalent to cancel button in dialogs.
Backspace	Delete the character before the cursor
Ctrl+BS	Delete the word before the cursor
Arrows --	
Left	Move the cursor to the left one character
Ctrl+Left	Move the cursor backward one word (Word delimiters are settable, see " Customizing NEdit ", and " X Resources ")

Nirvana Editor (NEdit) Help Documentation

Right	Move the cursor to the right one character
Ctrl+Right	Move the cursor forward one word
Up	Move the cursor up one line
Ctrl+Up	Move the cursor up one paragraph. (Paragraphs are delimited by blank lines)
Down	Move the cursor down one line.
Ctrl+Down	Move the cursor down one paragraph.
Ctrl+Return	Return with automatic indent, regardless of the setting of Auto Indent.
Shift+Return	Return without automatic indent, regardless of the setting of Auto Indent.
Ctrl+Tab	Insert an ASCII tab character, without processing emulated tabs.
Alt+Ctrl+<c>	Insert the control-code equivalent of a key <c>
Ctrl+/	Select everything (same as Select All menu item or ^A)
Ctrl+\	Unselect
Ctrl+U	Delete to start of line

PC Standard Keyboard

Ctrl+Insert	Copy the primary selection to the clipboard (same as Copy menu item or ^C) for compatibility with Motif standard key binding
Shift+Ctrl+Insert	Copy the primary selection to the cursor location.
Delete	Delete the character before the cursor. (Can be configured to delete the character after the cursor, see " Customizing NEdit ", and " X Resources ")
Ctrl+Delete	Delete to end of line.
Shift+Delete	Cut, remove the currently selected text and place it in the clipboard. (same as Cut menu item or ^X) for compatibility with Motif standard key binding
Shift+Ctrl+Delete	Cut the primary selection to the cursor location.
Home	Move the cursor to the beginning of the

Nirvana Editor (NEdit) Help Documentation

	line
Ctrl+Home	Move the cursor to the beginning of the file
End	Move the cursor to the end of the line
Ctrl+End	Move the cursor to the end of the file
PageUp	Scroll and move the cursor up by one page.
Ctrl+PageUp	Scroll and move the cursor left by one page.
PageDown	Scroll and move the cursor down by one page.
Ctrl+PageDown	Scroll and move the cursor right by one page.
F10	Make the menu bar active for keyboard input (Arrow Keys, Return, Escape, and the Space Bar)

Specialty Keyboards

On machines with different styles of keyboards, generally, text editing actions are properly matched to the labeled keys, such as Remove, Next–screen, etc.. If you prefer different key bindings, see the section titled "[Key Binding](#)" under the Customizing heading in the Help menu.

Shifting and Filling

Shift Left, Shift Right

While shifting blocks of text is most important for programmers (See Features for Programming), it is also useful for other tasks, such as creating indented paragraphs.

To shift a block of text one tab stop to the right, select the text, then choose Shift Right from the Edit menu. Note that the accelerator keys for these menu items are Ctrl+9 and Ctrl+0, which correspond to the right and left parenthesis on most keyboards. Remember them as adjusting the text in the direction pointed to by the parenthesis character. Holding the Shift key while selecting either Shift Left or Shift Right will shift the text by one character.

It is also possible to shift blocks of text by selecting the text rectangularly, and dragging it left or right (and up or down as well). Using a rectangular selection also causes tabs within the selection to be recalculated and substituted, such that the non–whitespace characters remain stationary with respect to the selection.

Filling

Text filling using the Fill Paragraph command in the Edit menu is one of the most important concepts in NEdit. And it will be well worth your while to understand how to use it properly.

In plain text files, unlike word-processor files, there is no way to tell which lines are continuations of other lines, and which lines are meant to be separate, because there is no distinction in meaning between newline characters which separate lines in a paragraph, and ones which separate paragraphs from other text. This makes it impossible for a text editor like NEdit to tell parts of the text which belong together as a paragraph from carefully arranged individual lines.

In continuous wrap mode (Preferences → Wrap → Continuous), lines automatically wrap and unwrap themselves to line up properly at the right margin. In this mode, you simply omit the newlines within paragraphs and let NEdit make the line breaks as needed. Unfortunately, continuous wrap mode is not appropriate in the majority of situations, because files with extremely long lines are not common under Unix and may not be compatible with all tools, and because you can't achieve effects like indented sections, columns, or program comments, and still take advantage of the automatic wrapping.

Without continuous wrapping, paragraph filling is not entirely automatic. Auto-Newline wrapping keeps paragraphs lined up as you type, but once entered, NEdit can no longer distinguish newlines which join wrapped text, and newlines which must be preserved. Therefore, editing in the middle of a paragraph will often leave the right margin messy and uneven.

Since NEdit can't act automatically to keep your text lined up, you need to tell it explicitly where to operate, and that is what Fill Paragraph is for. It arranges lines to fill the space between two margins, wrapping the lines neatly at word boundaries. Normally, the left margin for filling is inferred from the text being filled. The first line of each paragraph is considered special, and its left indentation is maintained separately from the remaining lines (for leading indents, bullet points, numbered paragraphs, etc.). Otherwise, the left margin is determined by the furthest left non-whitespace character. The right margin is either the Wrap Margin, set in the preferences menu (by default, the right edge of the window), or can also be chosen on the fly by using a rectangular selection (see below).

There are three ways to use Fill Paragraph. The simplest is, while you are typing text, and there is no selection, simply select Fill Paragraph (or type Ctrl+J), and NEdit will arrange the text in the paragraph adjacent to the cursor. A paragraph, in this case, means an area of text delimited by blank lines.

The second way to use Fill Paragraph is with a selection. If you select a range of text and then chose Fill Paragraph, all of the text in the selection will be filled. Again, continuous text between blank lines is interpreted as paragraphs and filled individually, respecting leading indents and blank lines.

The third, and most versatile, way to use Fill Paragraph is with a rectangular selection. Fill Paragraph treats rectangular selections differently from other commands. Instead of simply filling the text inside the rectangular selection, NEdit interprets the right edge of the selection as the requested wrap margin. Text to the left of the selection is not disturbed (the usual interpretation of a rectangular selection), but text to the right of the selection is included in the operation and is pulled in to the selected region. This method enables you to fill text to an arbitrary right margin, without going back and forth to the wrap-margin dialog, as well as to exclude text to the left of the selection such as comment bars or other text columns.

File Format

While plain-text is probably the simplest and most interchangeable file format in the computer world, there is still variation in what plain-text means from system to system. Plain-text files can differ in character set, line termination, and wrapping.

Nirvana Editor (NEdit) Help Documentation

While character set differences are the most obvious and pose the most challenge to portability, they affect NEdit only indirectly via the same font and localization mechanisms common to all X applications. If your system is set up properly, you will probably never see character-set related problems in NEdit. NEdit can not display Unicode text files, or any multi-byte character set.

The primary difference between an MS DOS format file and a Unix format file, is how the lines are terminated. Unix uses a single newline character. MS DOS uses a carriage-return and a newline. NEdit can read and write both file formats, but internally, it uses the single character Unix standard. NEdit auto-detects MS DOS format files based on the line termination at the start of the file. Files are judged to be DOS format if all of the first five line terminators, within a maximum range, are DOS-style. To change the format in which NEdit writes a file from DOS to Unix or visa versa, use the Save As... command and check or un-check the MS DOS Format button.

Wrapping within text files can vary among individual users, as well as from system to system. Both Windows and MacOS make frequent use of plain text files with no implicit right margin. In these files, wrapping is determined by the tool which displays them. Files of this style also exist on Unix systems, despite the fact that they are not supported by all Unix utilities. To display this kind of file properly in NEdit, you have to select the wrap style called Continuous. Wrapping modes are discussed in the sections: Customizing -> Preferences, and Basic Operation -> Shifting and Filling.

The last and most minute of format differences is the terminating newline. Some Unix compilers and utilities require a final terminating newline on all files they read and fail in various ways on files which do not have it. Vi and approximately half of Unix editors enforce the terminating newline on all files that they write; Emacs does not enforce this rule. Users are divided on which is best. NEdit makes the final terminating newline optional (Preferences -> Default Settings -> Terminate with Line Break on Save).

Features for Programming

Programming with NEdit

Though general in appearance, NEdit has many features intended specifically for programmers. Major programming-related topics are listed in separate sections under the heading: "Features for Programming": Syntax Highlighting, Tabs/Emulated Tabs, Finding Declarations (ctags), Calltips, and Auto/Smart Indent. Minor topics related to programming are discussed below:

Language Modes

When NEdit initially reads a file, it attempts to determine whether the file is in one of the computer languages that it knows about. Knowing what language a file is written in allows NEdit to assign highlight patterns and smart indent macros, and to set language specific preferences like word delimiters, tab emulation, and auto-indent. Language mode can be recognized from both the file name and from the first 200 characters of content. Language mode recognition and language-specific preferences are configured in: Preferences -> Default Settings -> Language Modes....

You can set the language mode manually for a window, by selecting it from the menu: Preferences -> Language Modes.

Backlighting [EXPERIMENTAL]

NEdit can be made to set the background color of particular classes of characters to allow easy identification of those characters. This is particularly useful if you need to be able to distinguish between tabs and spaces in a file where the difference is important. The colors used for backlighting are specified by a resource, "nedit*backlightCharTypes". You can turn backlighting on and off through the Preferences -> Apply Backlighting menu entry.

If you prefer to have backlighting turned on for all new windows, use the Preferences -> Default Settings -> Apply Backlighting menu entry. This settings can be saved along with other preferences using Preferences -> Save Defaults.

Important: In future versions of NEdit, the backlighting feature will be extended and reworked such that it becomes easier to configure. The current way of controlling it through a resource is generally considered to be below NEdit's usability standards. These future changes are likely to be incompatible with the current format of the "nedit*backlightCharTypes" resource, though. Therefore, it is expected that there will be no automatic migration path for users who customize the resource.

Line Numbers

To find a particular line in a source file by line number, choose Goto Line #... from the Search menu. You can also directly select the line number text in the compiler message in the terminal emulator window (xterm, decterm, winterm, etc.) where you ran the compiler, and choose Goto Selected from the Search menu.

To find out the line number of a particular line in your file, turn on Statistics Line in the Preferences menu and position the insertion point anywhere on the line. The statistics line continuously updates the line number of the line containing the cursor.

To go to a specific column on a given line, choose Goto Line #... from the Search menu and enter a line number and a column number separated by a comma. (e.g. Enter "100,12" for line 100 column 12.) If you want to go to a column on the current line just leave out the line number. (e.g. Enter ",45" to go the column 45 on the current line.)

Matching Parentheses

To help you inspect nested parentheses, brackets, braces, quotes, and other characters, NEdit has both an automatic parenthesis matching mode, and a Goto Matching command. Automatic parenthesis matching is activated when you type, or move the insertion cursor after a parenthesis, bracket, or brace. It momentarily highlights either the opposite character ('Delimiter') or the entire expression ('Range') when the opposite character is visible in the window. To find a matching character anywhere in the file, select it or position the cursor after it, and choose Goto Matching from the Search menu. If the character matches itself, such as a quote or slash, select the first character of the pair. NEdit will match {, (, [, <, ", ', `, /, and \. Holding the Shift key while typing the accelerator key (Shift+Ctrl+M, by default), will select all of the text between the matching characters.

When syntax highlighting is enabled, the matching routines can optionally make use of the syntax information for improved accuracy. In that case, a brace inside a highlighted string will not match a brace inside a comment, for instance.

Opening Included Files

The Open Selected command in the File menu understands the C preprocessor's #include syntax, so selecting an #include line and invoking Open Selected will generally find the file referred to, unless doing so depends on the settings of compiler switches or other information not available to NEdit.

Interface to Programming Tools

Integrated software development environments such as SGI's CaseVision and Centerline Software's Code Center, can be interfaced directly with NEdit via the client server interface. These tools allow you to click directly on compiler and runtime error messages and request NEdit to open files, and select lines of interest. The easiest method is usually to use the tool's interface for character-based editors like vi, to invoke nc, but programmatic interfaces can also be derived using the source code for nc.

There are also some simple compile/review, grep, ctree, and ctags browsers available in the NEdit contrib directory on ftp.nedit.org.

Tabs/Emulated Tabs

Changing the Tab Distance

Tabs are important for programming in languages which use indentation to show nesting, as short-hand for producing white-space for leading indents. As a programmer, you have to decide how to use indentation, and how or whether tab characters map to your indentation scheme.

Ideally, tab characters map directly to the amount of indent that you use to distinguish nesting levels in your code. Unfortunately, the Unix standard for interpretation of tab characters is eight characters (probably dating

back to mechanical capabilities of the original teletype), which is usually too coarse for a single indent.

Most text editors, NEdit included, allow you to change the interpretation of the tab character, and many programmers take advantage of this, and set their tabs to 3 or 4 characters to match their programming style. In NEdit you set the hardware tab distance in Preferences → Tabs... for the current window, or Preferences → Default Settings → Tabs... (general), or Preferences → Default Settings → Language Modes... (language-specific) to change the defaults for future windows.

Changing the meaning of the tab character makes programming much easier while you're in the editor, but can cause you headaches outside of the editor, because there is no way to pass along the tab setting as part of a plain-text file. All of the other tools which display, print, and otherwise process your source code have to be made aware of how the tabs are set, and must be able to handle the change. Non-standard tabs can also confuse other programmers, or make editing your code difficult for them if their text editors don't support changes in tab distance.

Emulated Tabs

An alternative to changing the interpretation of the tab character is tab emulation. In the Tabs... dialog(s), turning on Emulated Tabs causes the Tab key to insert the correct number of spaces and/or tabs to bring the cursor the next emulated tab stop, as if tabs were set at the emulated tab distance rather than the hardware tab distance. Backspacing immediately after entering an emulated tab will delete the fictitious tab as a unit, but as soon as you move the cursor away from the spot, NEdit will forget that the collection of spaces and tabs is a tab, and will treat it as separate characters. To enter a real tab character with "Emulate Tabs" turned on, use Ctrl+Tab.

It is also possible to tell NEdit not to insert ANY tab characters at all in the course of processing emulated tabs, and in shifting and rectangular insertion/deletion operations, for programmers who worry about the misinterpretation of tab characters on other systems.

Auto/Smart Indent

Programmers who use structured languages usually require some form of automatic indent, so that they don't have to continually re-type the sequences of tabs and/or spaces needed to maintain lengthy running indents. NEdit therefore offers "smart" indent, in addition to the traditional automatic indent which simply lines up the cursor position with the previous line.

Smart Indent

Smart indent macros are only available by default for C and C++, and while these can easily be configured for different default indentation distances, they may not conform to everyone's exact C programming style. Smart indent is programmed in terms of macros in the NEdit macro language which can be entered in: Preferences → Default Settings → Indent → Program Smart Indent. Hooks are provided for intervening at the point that a newline is entered, either via the user pressing the Enter key, or through auto-wrapping; and for arbitrary type-in to act on specific characters typed.

To type a newline character without invoking smart-indent when operating in smart-indent mode, hold the Shift key while pressing the Return or Enter key.

Auto-Indent

With Indent set to Auto (the default), NEdit keeps a running indent. When you press the Return or Enter key, spaces and tabs are inserted to line up the insert point under the start of the previous line.

Regardless of indent-mode, Ctrl+Return always does the automatic indent; Shift+Return always does a return without indent.

Block Indentation Adjustment

The Shift Left and Shift Right commands as well as rectangular dragging can be used to adjust the indentation for several lines at once. To shift a block of text one character to the right, select the text, then choose Shift Right from the Edit menu. Note that the accelerator keys for these menu items are Ctrl+9 and Ctrl+0, which correspond to the right and left parenthesis on most keyboards. Remember them as adjusting the text in the direction pointed to by the parenthesis character. Holding the Shift key while selecting either Shift Left or Shift Right will shift the text by one tab stop (or by one emulated tab stop if tab emulation is turned on). The help section "Shifting and Filling" under "Basic Operation" has details.

Syntax Highlighting

Syntax Highlighting means using colors and fonts to help distinguish language elements in programming languages and other types of structured files. Programmers use syntax highlighting to understand code faster and better, and to spot many kinds of syntax errors more quickly.

To use syntax highlighting in NEdit, select Highlight Syntax in the Preferences menu. If NEdit recognizes the computer language that you are using, and highlighting rules (patterns) are available for that language, it will highlight your text, and maintain the highlighting, automatically, as you type.

If NEdit doesn't correctly recognize the type of the file you are editing, you can manually select a language mode from Language Modes in the Preferences menu. You can also program the method that NEdit uses to recognize language modes in Preferences -> Default Settings -> Language Modes....

If no highlighting patterns are available for the language that you want to use, you can create new patterns relatively quickly. The Help section "[Highlighting Patterns](#)" under "Customizing", has details.

If you are satisfied with what NEdit is highlighting, but would like it to use different colors or fonts, you can change these by selecting Preferences -> Default Settings -> Syntax Highlighting -> Text Drawing Styles. Highlighting patterns are connected with font and color information through a common set of styles so that colorings defined for one language will be similar across others, and patterns within the same language which are meant to appear identical can be changed in the same place. To understand which styles are used to highlight the language you are interested in, you may need to look at "[Highlighting Patterns](#)" section, as well.

Syntax highlighting is CPU intensive, and under some circumstances can affect NEdit's responsiveness. If you have a particularly slow system, or work with very large files, you may not want to use it all of the time. Syntax highlighting introduces two kinds of delays. The first is an initial parsing delay, proportional to the size of the file. This delay is also incurred when pasting large sections of text, filtering text through shell commands, and other circumstances involving changes to large amounts of text. The second kind of delay happens when text which has not previously been visible is scrolled in to view. Depending on your system,

and the highlight patterns you are using, this may or may not be noticeable. A typing delay is also possible, but unlikely if you are only using the built-in patterns.

Finding Declarations (ctags)

NEdit can process tags files generated using the Unix ctags command or the Exuberant Ctags program. Ctags creates index files correlating names of functions and declarations with their locations in C, Fortran, or Pascal source code files. (See the ctags manual page for more information). Ctags produces a file called "tags" which can be loaded by NEdit. NEdit can manage any number of tags files simultaneously. Tag collisions are handled with a popup menu to let the user decide which tag to use. In 'Smart' mode NEdit will automatically choose the desired tag based on the scope of the file or module. Once loaded, the information in the tags file enables NEdit to go directly to the declaration of a highlighted function or data structure name with a single command. To load a tags file, select "Load Tags File" from the File menu and choose a tags file to load, or specify the name of the tags file on the NEdit command line:

```
nedit -tags tags
```

NEdit can also be set to load a tags file automatically when it starts up. Setting the X resource nedit.tagFile to the name of a tag file tells NEdit to look for that file at startup time (see "[Customizing NEdit](#)"). The file name can be either a complete path name, in which case NEdit will always load the same tags file, or a file name without a path or with a relative path, in which case NEdit will load it starting from the current directory. The second option allows you to have different tags files for different projects, each automatically loaded depending on the directory you're in when you start NEdit. Setting the name to "tags" is an obvious choice since this is the name that ctags uses. NEdit normally evaluates relative path tag file specifications every time a file is opened. All accessible tag files are loaded at this time. To disable the automatic loading of tag files specified as relative paths, set the X resource nedit.alwaysCheckRelativeTagsSpecs to False.

To unload a tags file, select "Un-load Tags File" from the File menu and choose from the list of tags files. NEdit will keep track of tags file updates by checking the timestamp on the files, and automatically update the tags cache.

To find the definition of a function or data structure once a tags file is loaded, select the name anywhere it appears in your program (see "[Selecting Text](#)") and choose "Find Definition" from the Search menu.

Calltips

Calltips are little yellow boxes that pop up to remind you what the arguments and return type of a function are. More generally, they're a UI mechanism to present a small amount of crucial information in a prominent location. To display a calltip, select some text and choose "Show Calltip" from the Search menu. To kill a displayed calltip, hit Esc.

Calltips get their information from one of two places — either a tags file (see "[Finding Declarations \(ctags\)](#)") or a calltips file. First, any loaded calltips files are searched for a definition, and if nothing is found then the tags database is searched. If a tag is found that matches the highlighted text then a calltip is displayed with the first few lines of the definition — usually enough to show you what the arguments of a function are.

You can load a calltips file by using choosing "Load Calltips File" from the File menu. You can unload a

Nirvana Editor (NEdit) Help Documentation

calltips file by selecting it from the "Unload Calltips File" submenu of the File menu. You can also choose one or more default calltips files to be loaded for each language mode using the "Default calltips file(s)" field of the Language Modes dialog.

The calltips file format is very simple. calltips files are organized in blocks separated by blank lines. The first line of the block is the key, which is the word that is matched when a calltip is requested. The rest of the block is displayed as the calltip.

Almost any text at all can appear in a calltip key or a calltip. There are no special characters that need to be escaped. The only issues to note are that trailing whitespace is ignored, and you cannot have a blank line inside a calltip. (Use a single period instead — it'll be nearly invisible.) You should also avoid calltip keys that begin and end with '*' characters, since those are used to mark special blocks.

There are five special block types—comment, include, language, alias, and version—which are distinguished by their first lines, "* comment *", "* include *", "* language *", "* alias *", and "* version *" respectively (without quotes).

Comment blocks are ignored when reading calltips files.

Include blocks specify additional calltips files to load, one per line. The ~ character can be used for your \$HOME directory, but other shell shortcuts like * and ? can't be used. Include blocks allow you to make a calltips file for your project that includes, say, the calltips files for C, Motif, and Xt.

Language blocks specify which language mode the calltips should be used with. When a calltip is requested it won't match tips from languages other than the current language mode. Language blocks only affect the tips listed after the block.

Alias blocks allow a calltip to have multiple keys. The first line of the block is the key for the calltip to be displayed, and the rest of the lines are additional keys, one per line, that should also show the calltip.

Version blocks are ignored for the time being.

You can use calltips in your own macros using the calltip() and kill_calltip() macro subroutines and the \$calltip_ID macro variable. See the [Macro Subroutines](#) section for details.

Regular Expressions

Basic Regular Expression Syntax

Regular expressions (regex's) are useful as a way to match inexact sequences of characters. They can be used in the 'Find...' and 'Replace...' search dialogs and are at the core of Color Syntax Highlighting patterns. To specify a regular expression in a search dialog, simply click on the 'Regular Expression' radio button in the dialog.

A regex is a specification of a pattern to be matched in the searched text. This pattern consists of a sequence of tokens, each being able to match a single character or a sequence of characters in the text, or assert that a specific position within the text has been reached (the latter is called an anchor.) Tokens (also called atoms) can be modified by adding one of a number of special quantifier tokens immediately after the token. A quantifier token specifies how many times the previous token must be matched (see below.)

Tokens can be grouped together using one of a number of grouping constructs, the most common being plain parentheses. Tokens that are grouped in this way are also collectively considered to be a regex atom, since this new larger atom may also be modified by a quantifier.

A regex can also be organized into a list of alternatives by separating each alternative with pipe characters, '|'. This is called alternation. A match will be attempted for each alternative listed, in the order specified, until a match results or the list of alternatives is exhausted (see [Alternation](#) section below.)

The 'Any' Character

If a dot ('.') appears in a regex, it means to match any character exactly once. By default, dot will not match a newline character, but this behavior can be changed (see help topic [Parenthetical Constructs](#), under the heading, Matching Newlines).

Character Classes

A character class, or range, matches exactly one character of text, but the candidates for matching are limited to those specified by the class. Classes come in two flavors as described below:

```
[...] Regular class, match only characters listed.  
[^...] Negated class, match only characters NOT listed.
```

As with the dot token, by default negated character classes do not match newline, but can be made to do so.

The characters that are considered special within a class specification are different than the rest of regex syntax as follows. If the first character in a class is the '\' character (second character if the first character is '^') it is a literal character and part of the class character set. This also applies if the first or last character is '-'. Outside of these rules, two characters separated by '-' form a character range which includes all the characters between the two characters as well. For example, '[^f-j]' is the same as '[^fghij]' and means to match any character that is not 'f', 'g', 'h', 'i', or 'j'.

Anchors

Anchors are assertions that you are at a very specific position within the search text. NEdit regular expressions support the following anchor tokens:

```

^   Beginning of line
$   End of line
<   Left word boundary
>   Right word boundary
\b  Not a word boundary

```

Note that the `\B` token ensures that the left and right characters are both delimiter characters, or that both left and right characters are non-delimiter characters. Currently word anchors check only one character, e.g. the left word anchor `<` only asserts that the left character is a word delimiter character. Similarly the right word anchor checks the right character.

Quantifiers

Quantifiers specify how many times the previous regular expression atom may be matched in the search text. Some quantifiers can produce a large performance penalty, and can in some instances completely lock up NEdit. To prevent this, avoid nested quantifiers, especially those of the maximal matching type (see below.)

The following quantifiers are maximal matching, or "greedy", in that they match as much text as possible.

```

*   Match zero or more
+   Match one or more
?   Match zero or one

```

The following quantifiers are minimal matching, or "lazy", in that they match as little text as possible.

```

*?  Match zero or more
+?  Match one or more
??  Match zero or one

```

One final quantifier is the counting quantifier, or brace quantifier. It takes the following basic form:

```

{min,max} Match from `min' to `max' times the
           previous regular expression atom.

```

If ``min'` is omitted, it is assumed to be zero. If ``max'` is omitted, it is assumed to be infinity. Whether specified or assumed, ``min'` must be less than or equal to ``max'`. Note that both ``min'` and ``max'` are limited to 65535. If both are omitted, then the construct is the same as ``*'`. Note that ``{,}'` and ``{ }'` are both valid brace constructs. A single number appearing without a comma, e.g. ``{3}'` is short for the ``{min,min}'` construct, or to match exactly ``min'` number of times.

The quantifiers ``{1}'` and ``{1,1}'` are accepted by the syntax, but are optimized away since they mean to match exactly once, which is redundant information. Also, for efficiency, certain combinations of ``min'` and ``max'` are converted to either ``*'`, ``+'`, or ``?'` as follows:

```

{}  {,}  {0,}  *
{1,}  +
{,1}  {0,1}  ?

```

Note that {0} and {0,0} are meaningless and will generate an error message at regular expression compile time.

Brace quantifiers can also be "lazy". For example {2,5}? would try to match 2 times if possible, and will only match 3, 4, or 5 times if that is what is necessary to achieve an overall match.

Alternation

A series of alternative patterns to match can be specified by separating them with vertical pipes, '|'. An example of alternation would be `a|be|sea`. This will match `a`, or `be`, or `sea`. Each alternative can be an arbitrarily complex regular expression. The alternatives are attempted in the order specified. An empty alternative can be specified if desired, e.g. `a|b|`. Since an empty alternative can match nothingness (the empty string), this guarantees that the expression will match.

Comments

Comments are of the form `(?!#<comment text>)` and can be inserted anywhere and have no effect on the execution of the regular expression. They can be handy for documenting very complex regular expressions. Note that a comment begins with `(!#` and ends at the first occurrence of an ending parenthesis, or the end of the regular expression... period. Comments do not recognize any escape sequences.

Metacharacters

Escaping Metacharacters

In a regular expression (regex), most ordinary characters match themselves. For example, `ab%` would match anywhere `a` followed by `b` followed by `%` appeared in the text. Other characters don't match themselves, but are metacharacters. For example, backslash is a special metacharacter which 'escapes' or changes the meaning of the character following it. Thus, to match a literal backslash would require a regular expression to have two backslashes in sequence. NEdit provides the following escape sequences so that metacharacters that are used by the regex syntax can be specified as ordinary characters.

```
\( \) \- \[ \] \< \> \{ \}
\. \| \^ \$ \* \+ \? \& \\\
```

Special Control Characters

There are some special characters that are difficult or impossible to type. Many of these characters can be constructed as a sort of metacharacter or sequence by preceding a literal character with a backslash. NEdit recognizes the following special character sequences:

```
\a alert (bell)
\b backspace
\e ASCII escape character (***)
\f form feed (new page)
\n newline
\r carriage return
\t horizontal tab
\v vertical tab
```

```
*** For environments that use the EBCDIC character set,
    when compiling NEdit set the EBCDIC_CHARSET compiler
    symbol to get the EBCDIC equivalent escape
    character.)
```

Octal and Hex Escape Sequences

Any ASCII (or EBCDIC) character, except null, can be specified by using either an octal escape or a hexadecimal escape, each beginning with `\0` or `\x` (or `\X`), respectively. For example, `\052` and `\X2A` both specify the ``*'` character. Escapes for null (`\00` or `\x0`) are not valid and will generate an error message. Also, any escape that exceeds `\0377` or `\xFF` will either cause an error or have any additional character(s) interpreted literally. For example, `\0777` will be interpreted as `\077` (a ``?'` character) followed by ``7'` since `\0777` is greater than `\0377`.

An invalid digit will also end an octal or hexadecimal escape. For example, `\091` will cause an error since ``9'` is not within an octal escape's range of allowable digits (0–7) and truncation before the ``9'` yields `\0` which is invalid.

Shortcut Escape Sequences

NEdit defines some escape sequences that are handy shortcuts for commonly used character classes.

```
\d  digits           0-9
\l  letters         a-z, A-Z, and locale dependent letters
\s  whitespace      \t, \r, \v, \f, and space
\w  word characters letters, digits, and underscore, `_'
```

`\D`, `\L`, `\S`, and `\W` are the same as the lowercase versions except that the resulting character class is negated. For example, `\d` is equivalent to ``[0-9]'`, while `\D` is equivalent to ``[^0-9]'`.

These escape sequences can also be used within a character class. For example, ``[l_]'` is the same as ``[a-zA-Z_]'`, extended with possible locale dependent letters. The escape sequences for special characters, and octal and hexadecimal escapes are also valid within a class.

Word Delimiter Tokens

Although not strictly a character class, the following escape sequences behave similarly to character classes:

```
\y  Word delimiter character
\Y  Not a word delimiter character
```

The `\y` token matches any single character that is one of the characters that NEdit recognizes as a word delimiter character, while the `\Y` token matches any character that is NOT a word delimiter character. Word delimiter characters are dynamic in nature, meaning that the user can change them through preference settings. For this reason, they must be handled differently by the regular expression engine. As a consequence of this, `\y` and `\Y` can not be used within a character class specification.

Parentetical Constructs

Capturing Parentheses

Capturing Parentheses are of the form ``(<regex>`` and can be used to group arbitrarily complex regular expressions. Parentheses can be nested, but the total number of parentheses, nested or otherwise, is limited to 50 pairs. The text that is matched by the regular expression between a matched set of parentheses is captured and available for text substitutions and backreferences (see below.) Capturing parentheses carry a fairly high overhead both in terms of memory used and execution speed, especially if quantified by ``*`` or ``+``.

Non-Capturing Parentheses

Non-Capturing Parentheses are of the form ``(?:<regex>`` and facilitate grouping only and do not incur the overhead of normal capturing parentheses. They should not be counted when determining numbers for capturing parentheses which are used with backreferences and substitutions. Because of the limit on the number of capturing parentheses allowed in a regex, it is advisable to use non-capturing parentheses when possible.

Positive Look-Ahead

Positive look-ahead constructs are of the form ``(?:=<regex>`` and implement a zero width assertion of the enclosed regular expression. In other words, a match of the regular expression contained in the positive look-ahead construct is attempted. If it succeeds, control is passed to the next regular expression atom, but the text that was consumed by the positive look-ahead is first unmatched (backtracked) to the place in the text where the positive look-ahead was first encountered.

One application of positive look-ahead is the manual implementation of a first character discrimination optimization. You can include a positive look-ahead that contains a character class which lists every character that the following (potentially complex) regular expression could possibly start with. This will quickly filter out match attempts that can not possibly succeed.

Negative Look-Ahead

Negative look-ahead takes the form ``(?:!<regex>`` and is exactly the same as positive look-ahead except that the enclosed regular expression must NOT match. This can be particularly useful when you have an expression that is general, and you want to exclude some special cases. Simply precede the general expression with a negative look-ahead that covers the special cases that need to be filtered out.

Positive Look-Behind

Positive look-behind constructs are of the form ``(?:<=<regex>`` and implement a zero width assertion of the enclosed regular expression in front of the current matching position. It is similar to a positive look-ahead assertion, except for the fact the the match is attempted on the text preceding the current position, possibly even in front of the start of the matching range of the entire regular expression.

A restriction on look-behind expressions is the fact that the expression must match a string of a bounded size. In other words, ``*``, ``+``, and ``{n,}`` quantifiers are not allowed inside the look-behind expression. Moreover, matching performance is sensitive to the difference between the upper and lower bound on the matching size. The smaller the difference, the better the performance. This is especially important for regular expressions used in highlight patterns.

Another (minor) restriction is the fact that look-**ahead** patterns, nor any construct that requires look-ahead information (such as word boundaries) are supported at the end of a look-behind pattern (no error is raised, but matching behaviour is unspecified). It is always possible to place these look-ahead patterns immediately after the look-behind pattern, where they will work as expected.

Positive look-behind has similar applications as positive look-ahead.

Negative Look-Behind

Negative look-behind takes the form ``(?<!<regex>)'` and is exactly the same as positive look-behind except that the enclosed regular expression must NOT match. The same restrictions apply.

Note however, that performance is even more sensitive to the distance between the size boundaries: a negative look-behind must not match for **any** possible size, so the matching engine must check **every** size.

Case Sensitivity

There are two parenthetical constructs that control case sensitivity:

```
(?i<regex>) Case insensitive; `AbcD' and `aBCd' are
equivalent.
```

```
(?I<regex>) Case sensitive; `AbcD' and `aBCd' are
different.
```

Regular expressions are case sensitive by default, that is, ``(?I<regex>)'` is assumed. All regular expression token types respond appropriately to case insensitivity including character classes and backreferences. There is some extra overhead involved when case insensitivity is in effect, but only to the extent of converting each character compared to lower case.

Matching Newlines

NEdit regular expressions by default handle the matching of newlines in a way that should seem natural for most editing tasks. There are situations, however, that require finer control over how newlines are matched by some regular expression tokens.

By default, NEdit regular expressions will NOT match a newline character for the following regex tokens: dot (`.`); a negated character class (`^[^...]`); and the following shortcuts for character classes:

```
`\d', `\D', `\l', `\L', `\s', `\S', `\w', `\W', `\Y'
```

The matching of newlines can be controlled for the ``.'` token, negated character classes, and the ``\s'` and ``\S'` shortcuts by using one of the following parenthetical constructs:

```
(?n<regex>) `.', `^[^...]', `\s', `\S' match newlines
```

```
(?N<regex>) `.', `^[^...]', `\s', `\S' don't match
newlines
```

``(?N<regex>)'` is the default behavior.

Notes on New Parenthetical Constructs

Except for plain parentheses, none of the parenthetical constructs capture text. If that is desired, the construct must be wrapped with capturing parentheses, e.g. `((?i<regex))'`.

All parenthetical constructs can be nested as deeply as desired, except for capturing parentheses which have a limit of 50 sets of parentheses, regardless of nesting level.

Back References

Backreferences allow you to match text captured by a set of capturing parenthesis at some later position in your regular expression. A backreference is specified using a single backslash followed by a single digit from 1 to 9 (example: `\3`). Backreferences have similar syntax to substitutions (see below), but are different from substitutions in that they appear within the regular expression, not the substitution string. The number specified with a backreference identifies which set of text capturing parentheses the backreference is associated with. The text that was most recently captured by these parentheses is used by the backreference to attempt a match. As with substitutions, open parentheses are counted from left to right beginning with 1. So the backreference `\3` will try to match another occurrence of the text most recently matched by the third set of capturing parentheses. As an example, the regular expression `(\d)\1` could match `'22'`, `'33'`, or `'00'`, but wouldn't match `'19'` or `'01'`.

A backreference must be associated with a parenthetical expression that is complete. The expression `(\w(\1))'` contains an invalid backreference since the first set of parentheses are not complete at the point where the backreference appears.

Substitution

Substitution strings are used to replace text matched by a set of capturing parentheses. The substitution string is mostly interpreted as ordinary text except as follows.

The escape sequences described above for special characters, and octal and hexadecimal escapes are treated the same way by a substitution string. When the substitution string contains the `&` character, NEdit will substitute the entire string that was matched by the `'Find...'` operation. Any of the first nine sub-expressions of the match string can also be inserted into the replacement string. This is done by inserting a `\` followed by a digit from 1 to 9 that represents the string matched by a parenthesized expression within the regular expression. These expressions are numbered left-to-right in order of their opening parentheses.

The capitalization of text inserted by `&` or `\1`, `\2`, ... `\9` can be altered by preceding them with `\U`, `\u`, `\L`, or `\l`. `\u` and `\l` change only the first character of the inserted entity, while `\U` and `\L` change the entire entity to upper or lower case, respectively.

Advanced Topics

Substitutions

Regular expression substitution can be used to program automatic editing operations. For example, the following are search and replace strings to find occurrences of the `'C'` language subroutine `'get_x'`, reverse the first and second parameters, add a third parameter of `NULL`, and change the name to `'new_get_x'`:

```
Search string:  `get_x *\(\ ([^ ,]*) , *([^\)]*)\)`  
Replace string: `new_get_x(\2, \1, NULL)'
```

Ambiguity

If a regular expression could match two different parts of the text, it will match the one which begins earliest. If both begin in the same place but match different lengths, or match the same length in different ways, life gets messier, as follows.

In general, the possibilities in a list of alternatives are considered in left-to-right order. The possibilities for `*`, `+`, and `?` are considered longest-first, nested constructs are considered from the outermost in, and concatenated constructs are considered leftmost-first. The match that will be chosen is the one that uses the earliest possibility in the first choice that has to be made. If there is more than one choice, the next will be made in the same manner (earliest possibility) subject to the decision on the first choice. And so forth.

For example, `(ab|a)b*c` could match `abc` in one of two ways. The first choice is between `ab` and `a`; since `ab` is earlier, and does lead to a successful overall match, it is chosen. Since the `b` is already spoken for, the `b*` must match its last possibility, the empty string, since it must respect the earlier choice.

In the particular case where no `|`s are present and there is only one `*`, `+`, or `?`, the net effect is that the longest possible match will be chosen. So `ab*`, presented with `xabbbby`, will match `abbbb`. Note that if `ab*` is tried against `xabyabbz`, it will match `ab` just after `x`, due to the begins-earliest rule. (In effect, the decision on where to start the match is the first choice to be made, hence subsequent choices must respect it even if this leads them to less-preferred alternatives.)

References

An excellent book on the care and feeding of regular expressions is

```
Mastering Regular Expressions, 2nd Edition  
Jeffrey E. F. Friedl  
2002, O'Reilly & Associates  
ISBN 0-596-00289-0
```

Example Regular Expressions

The following are regular expression examples which will match:

- An entire line.

```
^.*$
```

- Blank lines.

```
^$
```

- Whitespace on a line.

```
\s+
```


Nirvana Editor (NEdit) Help Documentation

- Whitespace across lines.

```
(?n\s+)
```

- Whitespace that spans at least two lines. Note minimal matching `*?' quantifier.

```
(?n\s*?\n\s*)
```

- IP address (not robust).

```
(?:\d{1,3}(?:\.\d{1,3}){3})
```

- Two character US Postal state abbreviations (includes territories).

```
[ACDF-IK-PR-W][A-Z]
```

- Web addresses.

```
(?:http://)?www\.\S+
```

- Case insensitive double words across line breaks.

```
(?i(?n<(\S+)\s+\1>))
```

- Upper case words with possible punctuation.

```
<[A-Z][^a-z\s]*>
```

Macro/Shell Extensions

Shell Commands and Filters

The Shell menu (Unix versions only) allows you to execute Unix shell commands from within NEdit. You can add items to the menu to extend NEdit's command set or to incorporate custom automatic editing features using shell commands or editing languages like awk and sed. To add items to the menu, select Preferences → Default Settings Customize Menus → Shell Menu. NEdit comes pre-configured with a few useful Unix commands like spell and sort, but we encourage you to add your own custom extensions.

Filter Selection... prompts you for a Unix command to use to process the currently selected text. The output from this command replaces the contents of the selection.

Execute Command... prompts you for a Unix command and replaces the current selection with the output of the command. If there is no selection, it deposits the output at the current insertion point. In the Shell Command field, the % character expands to the name (including directory path), and the # character expands to the current line number of the file in the window. To include a % or # character in the command, use %% or ##, respectively.

Execute Command Line uses the position of the cursor in the window to indicate a line to execute as a shell command line. The cursor may be positioned anywhere on the line. This command allows you to use an NEdit window as an editable command window for saving output and saving commands for re-execution. Note that the same character expansions described above in Execute Command also occur with this command.

The X resource called nedit.shell (See "[Customizing NEdit](#)") determines which Unix shell is used to execute commands. The default value for this resource is /bin/csh.

Learn/Replay

Selecting Learn Keystrokes from the Macro menu puts NEdit in learn mode. In learn mode, keystrokes and menu commands are recorded, to be played back later, using the Replay Keystrokes command, or pasted into a macro in the Macro Commands dialog of the Default Settings menu in Preferences.

Note that only keyboard and menu commands are recorded, not mouse clicks or mouse movements since these have no absolute point of reference, such as cursor or selection position. When you do a mouse-based operation in learn mode, NEdit will beep (repeatedly) to remind you that the operation was not recorded.

Learn mode is also the quickest and easiest method for writing macros. The dialog for creating macro commands contains a button labeled "Paste Learn / Replay Macro", which will deposit the last sequence learned into the body of the macro.

Repeating Actions and Learn/Replay Sequences

You can repeat the last (keyboard-based) command, or learn/replay sequence with the Repeat... command in the Macro menu. To repeat an action, first do the action (that is, insert a character, do a search, move the cursor), then select Repeat..., decide how or how many times you want it repeated, and click OK. For

example, to move down 30 lines through a file, you could type: <Down Arrow> Ctrl+, 29 <Return>. To repeat a learn/replay sequence, first learn it, then select Repeat..., click on Learn/Replay and how you want it repeated, then click OK.

If the commands you are repeating advance the cursor through the file, you can also repeat them within a range of characters, or from the current cursor position to the end of the file. To iterate over a range of characters, use the primary selection (drag the left mouse button over the text) to mark the range you want to operate on, and select "In Selection" in the Repeat dialog.

When using In "Selection" or "To End" with a learned sequence, try to do cursor movement as the last step in the sequence, since testing of the cursor position is only done at the end of the sequence execution. If you do cursor movement first, for example searching for a particular word then doing a modification, the position of the cursor won't be checked until the sequence has potentially gone far beyond the end of your desired range.

It's easy for a repeated command to get out of hand, and you can easily generate an infinite loop by using range iteration on a command which doesn't progress. To cancel a repeating command in progress, type Ctrl+. (period), or select Cancel Macro from the Macro menu.

Macro Language

Macros can be called from Macro menu commands, window background menu commands, within the smart-indent framework, from the autoload macro file and from the command line. Macro menu and window background menu commands are defined under Preferences -> Default Settings -> Customize Menus. Help on creating items in these menus can be found in the section, Help -> Customizing -> Preferences.

The autoload macro file is a file of macro commands and definitions which NEdit will automatically execute when it is first started. Its location is dependent on your environment:

- The default place for the file is '\$HOME/.nedit/autoload.nm',
- if the variable \$NEDIT_HOME is set in your environment it is located at '\$NEDIT_HOME/autoload.nm',
- if you are using old-style run control files (i.e. \$HOME/.nedit is a regular file) it is located in '\$HOME/.neditmacro'.

(For VMS, the file is in '\$NEDIT_HOME/autoload.nm' if \$NEDIT_HOME is set, in '\$SYS\$LOGIN:.neditmacro' otherwise.)

NEdit's macro language is a simple interpreter with integer arithmetic, dynamic strings, and C-style looping constructs (very similar to the procedural portion of the Unix awk program). From the macro language, you can call the same action routines which are bound to keyboard keys and menu items, as well additional subroutines for accessing and manipulating editor data, which are specific to the macro language (these are listed in the sections titled "[Macro Subroutines](#)", and "[Action Routines](#)").

Syntax

An NEdit macro language program consists of a list of statements, each terminated by a newline. Groups of statements which are executed together conditionally, such as the body of a loop, are surrounded by curly braces "{}".

Blank lines and comments are also allowed. Comments begin with a "#" and end with a newline, and can appear either on a line by themselves, or at the end of a statement.

Statements which are too long to fit on a single line may be split across several lines, by placing a backslash "\" character at the end of each line to be continued.

Data Types

The NEdit macro language recognizes only three data types, dynamic character strings, integer values and associative arrays. In general strings and integers can be used interchangeably. If a string represents an integer value, it can be used as an integer. Integers can be compared and concatenated with strings. Arrays may contain integers, strings, or arrays. Arrays are stored key/value pairs. Keys are always stored as strings.

Integer Constants

Integers are non-fractional numbers in the range of -2147483647 to 2147483647. Integer constants must be in decimal. For example:

```
a = -1
b = 1000
```

Character String Constants

Character string constants are enclosed in double quotes. For example:

```
a = "a string"
dialog("Hi there!", "Dismiss")
```

Strings may also include C-language style escape sequences:

```
\\ Backslash      \t Tab           \f Form feed
\" Double quote  \b Backspace    \a Alert
\n Newline       \r Carriage return \v Vertical tab
```

For example, to send output to the terminal from which NEdit was started, a newline character is necessary because, like printf, t_print requires explicit newlines, and also buffers its output on a per-line basis:

```
t_print("a = " a "\n")
```

Variables

Variable names must begin either with a letter (local variables), or a \$ (global variables). Beyond the first character, variables may also contain numbers and underscores `_'. Variables are called in to existence just by setting them (no explicit declarations are necessary).

Local variables are limited in scope to the subroutine (or menu item definition) in which they appear. Global variables are accessible from all routines, and their values persist beyond the call which created them, until reset.

Built-in Variables

NEdit has a number of permanently defined variables, which are used to access global editor information and information about the the window in which the macro is executing. These are listed along with the built in functions in the section titled "[Macro Subroutines](#)".

Functions and Subroutines

The syntax of a function or subroutine call is:

```
function_name(arg1, arg2, ...)
```

where arg1, arg2, etc. represent up to 9 argument values which are passed to the routine being called. A function or subroutine call can be on a line by itself, as above, or if it returns a value, can be invoked within a character or numeric expression:

```
a = fn1(b, c) + fn2(d)
dialog("fn3 says: " fn3())
```

Arguments are passed by value. This means that you can not return values via the argument list, only through the function value or indirectly through agreed-upon global variables.

Built-in Functions

NEdit has a wide range of built in functions which can be called from the macro language. These routines are divided into two classes, macro-language functions, and editor action routines. Editor action routines are more flexible, in that they may be called either from the macro language, or bound directly to keys via translation tables. They are also limited, however, in that they can not return values. Macro language routines can return values, but can not be bound to keys in translation tables.

Nearly all of the built-in subroutines operate on an implied window, which is initially the window from which the macro was started. To manipulate the contents of other windows, use the `focus_window` subroutine to change the focus to the ones you wish to modify. `focus_window` can also be used to iterate over all of the currently open windows, using the special keyword names, "last" and "next".

For backwards compatibility, hyphenated action routine names are allowed, and most of the existing action routines names which contain underscores have an equivalent version containing hyphens ('-') instead of underscores. Use of these names is discouraged. The macro parser resolves the ambiguity between '-' as the subtraction/negation operator, and - as part of an action routine name by assuming subtraction unless the symbol specifically matches an action routine name.

User Defined Functions

Users can define their own macro subroutines, using the `define` keyword:

```
define subroutine_name {
    < body of subroutine >
}
```

Macro definitions can not appear within other definitions, or within macro menu item definitions (usually they are found in the autoload macro file).

Nirvana Editor (NEdit) Help Documentation

The arguments with which a user-defined subroutine or function was invoked, are presented as \$1, \$2, ... , \$9. The number of arguments can be read from \$n_args.

To return a value from a subroutine, and/or to exit from the subroutine before the end of the subroutine body, use the return statement:

```
return <value to return>
```

Operators and Expressions

Operators have the same meaning and precedence that they do in C, except for ^, which raises a number to a power (y^x means y to the x power), rather than bitwise exclusive OR. The table below lists operators in decreasing order of precedence.

Operators	Associativity
()	
^	right to left
- ! ++ --	(unary)
* / %	left to right
+ -	left to right
> >= < <= == !=	left to right
&	left to right
	left to right
&&	left to right
	left to right
(concatenation)	left to right
= += -= *= /= %=, &= =	right to left

The order in which operands are evaluated in an expression is undefined, except for && and ||, which like C, evaluate operands left to right, but stop when further evaluation would no longer change the result.

Numerical Operators

The numeric operators supported by the NEdit macro language are listed below:

```
+ addition
- subtraction or negation
* multiplication
/ division
% modulo
^ power
& bitwise and
| bitwise or
```

Increment (++) and decrement (--) operators can also be appended or prepended to variables within an expression. Prepended increment/decrement operators act before the variable is evaluated. Appended increment/decrement operators act after the variable is evaluated.

Logical and Comparison Operators

Logical operations produce a result of 0 (for false) or 1 (for true). In a logical operation, any non-zero value is recognized to mean true. The logical and comparison operators allowed in the NEdit macro language are listed below:

```
&& logical and
```

```
|| logical or
! not
> greater
< less
>= greater or equal
<= less or equal
== equal (integers and/or strings)
!= not equal (integers and/or strings)
```

Character String Operators

The "operator" for concatenating two strings is the absence of an operator. Adjoining character strings with no operator in between means concatenation:

```
d = a b "string" c
t_print("the value of a is: " a)
```

Comparison between character strings is done with the == and != operators, (as with integers). There are a number of useful built-in routines for working with character strings, which are listed in the section called "[Macro Subroutines](#)".

Arrays and Array Operators

Arrays may contain either strings, integers, or other arrays. Arrays are associative, which means that they relate two pieces of information, the key and the value. The key is always a string; if you use an integer it is converted to a string.

To determine if a given key is in an array, use the 'in' keyword.

```
if ("6" in x)
  <body>
```

If the left side of the in keyword is an array, the result is true if every key in the left array is in the right array. Array values are not compared.

To iterate through all the keys of an array use the 'for' looping construct. Keys are not guaranteed in any particular order:

```
for (aKey in x)
  <body>
```

Elements can be removed from an array using the delete command:

```
delete x[3] # deletes element with key 3
delete x[] # deletes all elements
```

The number of elements in an array can be determined by referencing the array with no indices:

```
dialog("array x has " x[] " elements", "OK")
```

Arrays can be combined with some operators. All the following operators only compare the keys of the arrays.

```
result = x + y (Merge arrays)
```

Nirvana Editor (NEdit) Help Documentation

The 'result' is a new array containing keys from both x and y. If duplicates are present values from y are used.

```
result = x - y (Remove keys)
```

The 'result' is a new array containing all keys from x that are not in y.

```
result = x & y (Common keys)
```

The 'result' is a new array containing all keys which are in both x and y. The values from y are used.

```
result = x | y (Unique keys)
```

The 'result' is a new array containing keys which exist in either x or y, but not both.

When duplicate keys are encountered using the + and & operators, the values from the array on the right side of the operators are used for the result. All of the above operators are array only, meaning both the left and right sides of the operator must be arrays. The results are also arrays.

Array keys can also contain multiple dimensions:

```
x[1, 1, 1] = "string"
```

These are used in the expected way, e.g.:

```
for (i = 1; i < 3; i++)
{
    for (j = 1; j < 3; j++)
    {
        x[i, j] = k++
    }
}
```

gives the following array:

```
x[1, 1] = 0
x[1, 2] = 1
x[2, 1] = 2
x[2, 2] = 3
```

Internally all indices are part of one string, separated by the string \$sub_sep (ASCII 0x18). The first key in the above example is in fact

```
["1" $sub_sep "1"]
```

If you need to extract one of the keys, you can use split(), using \$sub_sep as the separator.

You can also check for the existence of multi-dimensional array by looking for \$sub_sep in the key.

Last, you need \$sub_sep if you want to use the 'in' keyword.

```
if ((1,2) in myArray)
{..}
```

doesn't work, but


```
if ("1" $sub_sep "2") in myArray
{..}
```

does work.

Looping and Conditionals

NEdit supports looping constructs: for and while, and conditional statements: if and else, with essentially the same syntax as C:

```
for (<init>, ...; <condition>; <increment>, ...) <body>
```

```
while (<condition>) <body>
```

```
if (<condition>) <body>
```

```
if (<condition>) <body> else <body>
```

<body>, as in C, can be a single statement, or a list of statements enclosed in curly braces ({}). <condition> is an expression which must evaluate to true for the statements in <body> to be executed. for loops may also contain initialization statements, <init>, executed once at the beginning of the loop, and increment/decrement statements (or any arbitrary statement), which are executed at the end of the loop, before the condition is evaluated again.

Examples:

```
for (i=0; i<100; i++)
  j = i * 2

for (i=0, j=20; i<20; i++, j--) {
  k = i * j
  t_print(i, j, k)
}

while (k > 0)
{
  k = k - 1
  t_print(k)
}

for (;;) {
  if (i-- < 1)
    break
}
```

Loops may contain break and continue statements. A break statement causes an exit from the innermost loop, a continue statement transfers control to the end of the loop.

Macro Subroutines

Built in Variables

These variables are read-only and can not be changed.

`$active_pane`

Index of the current pane.

`$auto_indent`

Contains the current preference for auto indent. Can be "off", "on" or "auto".

`$calltip_ID`

Equals the ID of the currently displayed calltip, or 0 if no calltip is being displayed.

`$cursor`

Position of the cursor in the current window.

`$column`

Column number of the cursor position in the current window.

`$display_width`

Width of the current pane in pixels.

`$em_tab_dist`

If tab emulation is turned on in the Tabs... dialog of the Preferences menu, value is the distance between emulated tab stops. If tab emulation is turned off, value is -1.

`$empty_array`

An array with no elements. This can be used to initialize an array to an empty state.

`$file_format`

Current newline format that the file will be saved with. Can be "unix", "dos" or "macintosh".

`$file_name`

Name of the file being edited in the current window, stripped of directory component.

`$file_path`

Directory component of file being edited in the current window.

`$font_name`

Contains the current plain text font name.

Nirvana Editor (NEdit) Help Documentation

`$font_name_bold`

Contains the current bold text font name.

`$font_name_bold_italic`

Contains the current bold-italic text font name.

`$font_name_italic`

Contains the current italic text font name.

`$highlight_syntax`

Whether syntax highlighting is turned on.

`$incremental_backup`

Contains 1 if incremental auto saving is on, otherwise 0.

`$incremental_search_line`

Has a value of 1 if the preference is selected to always show the incremental search line, otherwise 0.

`$language_mode`

Name of language mode set in the current window.

`$line`

Line number of the cursor position in the current window.

`$locked`

True if the file has been locked by the user.

`$make_backup_copy`

Has a value of 1 if original file is kept in a backup file on save, otherwise 0.

`$max_font_width`

The maximum font width of all the active styles. Syntax highlighting styles are only considered if syntax highlighting is turned on.

`$min_font_width`

The minimum font width of all the active styles. Syntax highlighting styles are only considered if syntax highlighting is turned on.

`$modified`

True if the file in the current window has been modified and the modifications have not yet been saved.

`$n_display_lines`

The number of lines visible in the currently active pane.

`$n_panes`

The number of panes in the current window.

`$overtypemode`

True if in Overtypemode.

`$readonly`

True if the file is read only.

`$selection_start, $selection_end`

Beginning and ending positions of the primary selection in the current window, or `-1` if there is no text selected in the current window.

`$selection_left, $selection_right`

Left and right character offsets of the rectangular (primary) selection in the current window, or `-1` if there is no selection or it is not rectangular.

`$servername`

Name of the current NEdit server.

`$show_line_numbers`

Whether line numbers are shown next to the text.

`$show_matching`

Contains the current preference for showing matching pairs, such as `"[]"` and `"{}"` pairs. Can be `"off"`, `"delimiter"`, or `"range"`.

`$match_syntax_based`

Whether pair matching should use syntax information, if available.

`$statistics_line`

Has a value of 1 if the statistics line is shown, otherwise 0.

`$sub_sep`

Contains the value of the array sub-script separation string.

`$tab_dist`

Nirvana Editor (NEdit) Help Documentation

The distance between tab stops for a hardware tab character, as set in the Tabs... dialog of the Preferences menu.

`$text_length`

The length of the text in the current window.

`$top_line`

The line number of the top line of the currently active pane.

`$use_tabs`

Whether the user is allowing the NEdit to insert tab characters to maintain spacing in tab emulation and rectangular dragging operations. (The setting of the "Use tab characters in padding and emulated tabs" button in the Tabs... dialog of the Preferences menu.)

`$wrap_margin`

The right margin in the current window for text wrapping and filling.

`$wrap_text`

The current wrap text mode. Values are "none", "auto" or "continuous".

Built-in Subroutines

`append_file(string, filename)`

Appends a string to a named file. Returns 1 on successful write, or 0 if unsuccessful.

`beep()`

Ring the bell.

`calltip("text_or_key" [, pos [, mode or position_modifier, ...]])`

Pops up a calltip. <pos> is an optional position in the buffer where the tip will be displayed. Passing -1 for <pos> is equivalent to not specifying a position, and it guarantees that the tip will appear on-screen somewhere even if the cursor is not. The upper-left corner of the calltip will appear below where the cursor would appear if it were at this position. <mode> is one of "tipText" (default), "tipKey", or "tagKey". "tipText" displays the text as-is, "tagKey" uses it as the key to look up a tag, then converts the tag to a calltip, and "tipKey" uses it as the key to look up a calltip, then falls back to "tagKey" behavior if that fails. You'll usually use "tipKey" or "tipText". Finally, you can modify the placement of the calltip relative to the cursor position (or <pos>) with one or more of these optional position modifiers: "center" aligns the center of the calltip with the position. "right" aligns the right edge of the calltip with the position. ("center" and "right" may not both be used.) "above" places the calltip above the position. "strict" does not allow the calltip to move from its position in order to avoid going off-screen or obscuring the cursor. Returns the ID of the calltip if it was found and/or displayed correctly, 0 otherwise.

`clipboard_to_string()`

Nirvana Editor (NEdit) Help Documentation

Returns the contents of the clipboard as a macro string. Returns empty string on error.

```
dialog( message, btn_1_label, btn_2_label, ... )
```

Pop up a dialog for querying and presenting information to the user. First argument is a string to show in the message area of the dialog. Up to eight additional optional arguments represent labels for buttons to appear along the bottom of the dialog. Returns the number of the button pressed (the first button is number 1), or 0 if the user closed the dialog via the window close box.

```
focus_window( window_name )
```

Sets the window on which subsequent macro commands operate. `window_name` can be either a fully qualified file name, or one of "last" for the last window created, or "next" for the next window in the chain from the currently focused window (the first window being the one returned from calling `focus_window("last")`). Returns the name of the newly-focused window, or an empty string if the requested window was not found.

```
get_character( position )
```

Returns the single character at the position indicated by the first argument to the routine from the current window.

```
get_range( start, end )
```

Returns the text between a starting and ending position from the current window.

```
get_selection()
```

Returns a string containing the text currently selected by the primary selection either from the current window (no keyword), or from anywhere on the screen (keyword "any").

```
getenv( name )
```

Gets the value of an environment variable.

```
kill_calltip( [calltip_ID] )
```

Kills any calltip that is being displayed in the window in which the macro is running. If there is no displayed calltip this does nothing. If a calltip ID is supplied then the calltip is killed only if its ID is `calltip_ID`.

```
length( string )
```

Returns the length of a string

```
list_dialog( message, text, btn_1_label, btn_2_label, ... )
```

Pop up a dialog for prompting the user to choose a line from the given text string. The first argument is a message string to be used as a title for the fixed text describing the list. The second string provides the list data: this is a text string in which list entries are separated by newline characters. Up to seven additional optional arguments represent labels for buttons to appear along the bottom of the dialog. Returns the line of text selected by the user as the function value (without any newline separator) or the empty string if none was selected, and number of the button pressed (the first button is number 1), in `$list_dialog_button`. If the user closes the dialog via the window close box, the function returns the empty string, and `$list_dialog_button`

returns 0.

`max(n1, n2, ...)`

Returns the maximum value of all of its arguments

`min(n1, n2, ...)`

Returns the minimum value of all of its arguments

`read_file(filename)`

Reads the contents of a text file into a string. On success, returns 1 in \$read_status, and the contents of the file as a string in the subroutine return value. On failure, returns the empty string "" and an 0 \$read_status.

`replace_in_string(string, search_for, replace_with [, type, "copy"])`

Replaces all occurrences of a search string in a string with a replacement string. Arguments are 1: string to search in, 2: string to search for, 3: replacement string. There are two optional arguments. One is a search type, either "literal", "case", "word", "caseWord", "regex", or "regexNoCase". The default search type is "literal". If the optional "copy" argument is specified, a copy of the input string is returned when no replacements were performed. By default an empty string ("") will be returned in this case. Returns a new string with all of the replacements done.

`replace_range(start, end, string)`

Replaces all of the text in the current window between two positions.

`replace_selection(string)`

Replaces the primary-selection selected text in the current window.

`replace_substring(string, start, end, replace_with)`

Replacing a substring between two positions in a string within another string.

`search(search_for, start [, search_type, wrap, direction])`

Searches silently in a window without dialogs, beeps, or changes to the selection. Arguments are: 1: string to search for, 2: starting position. Optional arguments may include the strings: "wrap" to make the search wrap around the beginning or end of the string, "backward" or "forward" to change the search direction ("forward" is the default), "literal", "case", "word", "caseWord", "regex", or "regexNoCase" to change the search type (default is "literal"). Returns the starting position of the match, or -1 if nothing matched. Also returns the ending position of the match in \$search_end.

`search_string(string, search_for, start [, search_type, direction])`

Built-in macro subroutine for searching a string. Arguments are 1: string to search in, 2: string to search for, 3: starting position. Optional arguments may include the strings: "wrap" to make the search wrap around the beginning or end of the string, "backward" or "forward" to change the search direction ("forward" is the default), "literal", "case", "word", "caseWord", "regex", or "regexNoCase" to change the search type (default is "literal"). Returns the starting position of the match, or -1 if nothing matched. Also returns the ending

position of the match in `$search_end`.

```
select( start, end )
```

Selects (with the primary selection) text in the current buffer between a starting and ending position.

```
select_rectangle( start, end, left, right )
```

Selects a rectangular area of text between a starting and ending position, and confined horizontally to characters displayed between positions "left", and "right".

```
set_cursor_pos( position )
```

Set the cursor position for the current window.

```
shell_command( command, input_string )
```

Executes a shell command, feeding it input from `input_string`. On completion, output from the command is returned as the function value, and the command's exit status is returned in the global variable `$shell_cmd_status`.

```
split(string, separation_string [, search_type])
```

Splits a string using the separator specified. Optionally the `search_type` argument can specify how the `separation_string` is interpreted. The default is "literal". The returned value is an array with keys beginning at 0.

```
string_dialog( message, btn_1_label, btn_2_label, ... )
```

Pops up a dialog prompting the user to enter information. The first argument is a string to show in the message area of the dialog. Up to nine additional optional arguments represent labels for buttons to appear along the bottom of the dialog. Returns the string entered by the user as the function value, and number of the button pressed (the first button is number 1), in `$string_dialog_button`. If the user closes the dialog via the window close box, the function returns the empty string, and `$string_dialog_button` returns 0.

```
string_compare(string1, string2 [, consider-case])
```

Compare two strings and return 0 if they are equal, -1 if `string1` is less than `string2` or 1 if `string1` is greater than `string2`. The values for the optional `consider-case` argument is either "case" or "nocase". The default is to do a case sensitive comparison.

```
string_to_clipboard( string )
```

Copy the contents of a macro string to the clipboard.

```
substring( string, start, end )
```

Returns the portion of a string between a starting and ending position.

```
t_print( string1, string2, ... )
```

Writes strings to the terminal (stdout) from which NEdit was started.

`tolower(string)`

Return an all lower-case version of string.

`toupper(string)`

Return an all upper-case version of string.

`valid_number(string)`

Returns 1 if the string can be converted to a number without error following the same rules that the implicit conversion would. Otherwise 0.

`write_file(string, filename)`

Writes a string (parameter 1) to a file named in parameter 2. Returns 1 on successful write, or 0 if unsuccessful.

Deprecated Functions

Some functions are included only for supporting legacy macros. You should not use any of these functions in any new macro you write. Among these are all action routines with hyphens in their names; use underscores instead ('find-dialog' -> 'find_dialog').

`match()`

DEPRECATED Use `select_to_matching()` instead.

Range Sets

The user can create range sets, identified by opaque integers. A range set contains ranges, defined by start and end positions in the text buffer. These ranges are adjusted when modifications are made to the text buffer: they shuffle around when characters are added or deleted. However, ranges within a set will coalesce if the characters between them are removed, or a new range is added to the set which bridges or overlaps others.

Using range sets allows non-contiguous bits of the text to be identified as a group.

Range sets can be assigned a background color: characters within a range of a range set will have the background color of the range set. If more than one rangeset includes a given character, its background color will be that of the most recently created range set which has a color defined.

Range sets must be created using the `rangeset_create()` function, which will return an identifier for the newly-created rangeset. This identifier is then passed to the other rangeset functions to manipulate the range set.

There is a limit to the number of range sets which can exist at any time – up to 63 in each text buffer. Care should be taken to destroy any rangesets which are no longer needed, by using the `rangeset_destroy()` function.

Warnings: A range set is manipulated ONLY through macro routines. Range sets can easily become very large, and may exceed the capacity of the running process. Coloring relies on proper color names or specifications (such as the "#rrggbb" hexadecimal digit strings), and appropriate hardware support. Behaviours set using `rangeset_set_mode()` are still experimental.

Range set read-only variables

`$rangeset_list`

array of active rangeset identifiers, with integer keys starting at 0, in the order the rangesets were defined.

Range set functions

`rangeset_create()`
`rangeset_create(n)`

Creates one or more new range sets. The first form creates a single range set and returns its identifier; if there are no range sets available it returns 0. The second form creates n new range sets, and returns an array of the range set identifiers with keys beginning at 0. If the requested number of range sets is not available it returns an empty array.

`rangeset_destroy(r)`
`rangeset_destroy(array)`

Deletes all information about a range set or a number of range sets. The first form destroys the range set identified by r. The second form should be passed an array of rangeset identifiers with keys beginning at 0 (i.e. the same form of array returned by `rangeset_create(n)`); it destroys all the range sets appearing in the array. If any of the range sets do not exist, the function continues without errors. Does not return a value.

`rangeset_add(r, [start, end])`
`rangeset_add(r, r0)`

Adds to the range set r. The first form adds the range identified by the current primary selection to the range set, unless start and end are defined, in which case the range they define is added. The second form adds all ranges in the range set r0 to the range set r. Returns the index of the newly-added range within the rangeset.

`rangeset_subtract(r, [start, end])`
`rangeset_subtract(r, r0)`

Removes from the range set r. The first form removes the range identified by the current primary selection from the range set, unless start and end are defined, in which case the range they define is removed. The second form removes all ranges in the range set r0 from the range set r. Does not return a value.

`rangeset_invert(r)`

Changes the range set r so that it contains all ranges not in r. Does not return a value.

`rangeset_get_by_name(name)`

Returns an array of active rangeset identifiers, with integer keys starting at 0, whose name matches name.

`rangeset_info(r)`

Returns an array containing information about the range set *r*. The array has the following keys: **defined** (whether a range set with identifier *r* is defined), **count** (the number of ranges in the range set), **color** (the current background color of the range set, an empty string if the range set has no color), **name** (the user supplied name of the range set, an empty string if the range set has no name), and **mode** (the name of the modify-response mode of the range set).

```
rangeset_range( r, [index] )
```

Returns details of a specific range in the range set *r*. The range is specified by *index*, which should be between 1 and *n* (inclusive), where *n* is the number of ranges in the range set. The return value is an array containing the keys **start** (the start position of the range) and **end** (the end position of the range). If *index* is not supplied, the region returned is the span of the entire range set (the region starting at the start of the first range and ending at the end of the last). If *index* is outside the correct range of values, the function returns an empty array.

```
rangeset_includes( r, pos )
```

Returns the index of the range in range set *r* which includes *pos*; returns 0 if *pos* is not contained in any of the ranges of *r*. This can also be used as a simple true/false function which returns true if *pos* is contained in the range set.

```
rangeset_set_color( r, color )
```

Attempts to apply the color as a background color to the ranges of *r*. If *color* is an empty string, removes the coloring of *r*. No check is made regarding the validity of *color*: if the color is invalid (a bad name, or not supported by the hardware) this has unpredictable effects.

```
rangeset_set_name( r, name )
```

Apply the name to the range set *r*.

```
rangeset_set_mode( r, type )
```

Changes the behaviour of the range set *r* when modifications to the text buffer occur. *type* can be one of the following: "maintain" (the default), "break", "include", "exclude", "ins_del" or "del_ins". (The differences are fairly subtle.)

Highlighting Information

The user can interrogate the current window to determine the color highlighting used on a particular piece of text. The following functions provide information on the highlighting pattern against which text at a particular position has been matched, its style, color and font attributes (whether the font is supposed to be bold and/or italic).

These macro functions permit macro writers to generate formatted output which allows NEdit highlighting to be reproduced. This is suitable for the generation of HTML or Postscript output, for example.

Note that if any of the functions is used while in Plain mode or while syntax highlighting is off, the behaviour is undefined.

Nirvana Editor (NEdit) Help Documentation

`get_pattern_by_name(pattern_name)`

Returns an array containing the pattern attributes for pattern 'pattern_name'. The elements in this array are:

- **style** — Highlight style name

If 'pattern_name' is invalid, an empty array is returned.

`get_pattern_at_pos(pos)`

Returns an array containing the pattern attributes of the character at position 'pos'. The elements in this array are:

- **pattern** — Highlight pattern name
- **style** — Highlight style name
- **extent** — The length in the text which uses the same highlighting pattern

The 'extent' value is measured from position 'pos' going right/down (forward in the file) only.

If 'pos' is invalid, an empty array is returned.

`get_style_by_name(style_name)`

Returns an array containing the style attributes for style 'style_name'. The elements in this array are:

- **bold** — '1' if style is bold, '0' otherwise
- **italic** — '1' if style is italic, '0' otherwise
- **color** — Name of the style's color
- **background** — Name of the background color, if any

The colors use the names specified in the color definitions for the style. These will either be names matching those the X server recognises, or RGB (red/green/black) specifications.

If 'style_name' is invalid, an empty array is returned.

`get_style_at_pos(pos)`

Returns an array containing the style attributes of the character at position 'pos'. The elements in this array are:

- **style** — Name of the highlight style
- **bold** — '1' if style is bold, '0' otherwise
- **italic** — '1' if style is italic, '0' otherwise
- **color** — Name of the style's color
- **rgb** — Color's RGB values ('#rrggbb')
- **background** — Name of the background color, if any
- **back_rgb** — Background color's RGB values ('#rrggbb')
- **extent** — The length in the text which uses the same highlight style

The colors use the names specified in the color definitions for the style. These will either be names matching those the X server recognises, or RGB specifications. The values for 'rgb' and 'back_rgb' contain the actual color values allocated by the X server for the window. If the X server cannot allocate the specified (named)

color exactly, the RGB values in these entries may not match the specified ones.

The 'extent' value is measured from position 'pos' going right/down (forward in the file) only.

If 'pos' is invalid, an empty array is returned.

Action Routines

All of the editing capabilities of NEdit are represented as a special type of subroutine, called an action routine, which can be invoked from both macros and translation table entries (see "[Key Binding](#)" in the Customizing section of the Help menu).

Actions Representing Menu Commands

File Menu -----	Search Menu -----
new()	find()
open()	find_dialog()
open_dialog()	find_again()
open_selected()	find_selection()
close()	replace()
save()	replace_dialog()
save_as()	replace_all()
save_as_dialog()	replace_in_selection()
revert_to_saved()	replace_again()
include_file()	goto_line_number()
include_file_dialog()	goto_line_number_dialog()
load_macro_file()	goto_selected()
load_macro_file_dialog()	mark()
load_tags_file()	mark_dialog()
load_tags_file_dialog()	goto_mark()
unload_tags_file()	goto_mark_dialog()
print()	goto_matching()
print_selection()	select_to_matching()
exit()	find_definition()
Edit Menu -----	Shell Menu -----
undo()	filter_selection_dialog()
redo()	filter_selection()
delete()	execute_command()
select_all()	execute_command_dialog()
shift_left()	execute_command_line()
shift_left_by_tab()	shell_menu_command()
shift_right()	
shift_right_by_tab()	Macro Menu -----
uppercase()	macro_menu_command()
lowercase()	repeat_macro()
fill_paragraph()	repeat_dialog()
control_code_dialog()	
	Windows Menu -----
	split_window()
	close_pane()

An action representing a menu command is named the same as its corresponding menu item except that all punctuation is removed, all letters are changed to lower case, and spaces are replaced with underscores. To present a dialog to ask the user for input, use the actions with the `_dialog` suffix. Actions without the `_dialog` suffix take the information from the routine's arguments (see below).

Menu Action Routine Arguments

Arguments are text strings enclosed in quotes. Below are the menu action routines which take arguments. Optional arguments are enclosed in [].

```
close( ["prompt" | "save" | "nosave" ] )

execute_command( shell-command )

filter_selection( shell-command )

find( search-string [, search-direction] [, search-type]
      [, search-wrap] )

find_again( [search-direction] [, search-wrap] )

find_definition( [tag-name] )

find_dialog( [search-direction] [, search-type]
              [, keep-dialog] )

find_selection( [search-direction] [, search-wrap]
                 [, non-regex-search-type] )

goto_line_number( [line-number] [, column-number] )

goto_mark( mark-letter )

include_file( filename )

load_tags_file( filename )

macro_menu_command( macro-menu-item-name )

mark( mark-letter )

open( filename )

replace( search-string, replace-string,
          [, search-direction] [, search-type] [, search-wrap] )

replace_again( [search-direction] [, search-wrap] )

replace_all( search-string, replace-string [, search-type] )

replace_dialog( [search-direction] [, search-type]
                 [, keep-dialog] )

replace_in_selection( search-string,
                       replace-string [, search-type] )
```

Nirvana Editor (NEdit) Help Documentation

save_as(*filename*)

shell_menu_command(*shell-menu-item-name*)

unload_tags_file(*filename*)

----- Some notes on argument types above -----

filename Path names are relative to the directory from which NEdit was started. Shell interpreted wildcards and `~' are not expanded.

keep-dialog Either "keep" or "nokeep".

mark-letter The mark command limits users to single letters. Inside of macros, numeric marks are allowed, which won't interfere with marks set by the user.

macro-menu-item-name Name of the command exactly as specified in the Macro Menu dialogs.

non-regex-search-type Either "literal", "case", "word", or "caseWord".

search-direction Either "forward" or "backward".

search-type Either "literal", "case", "word", "caseWord", "regex", or "regexNoCase".

search-wrap Either "wrap" or "nowrap".

shell-menu-item-name Name of the command exactly as specified in the Shell Menu dialogs.

Window Preferences Actions

set_auto_indent("off" | "on" | "smart")

Set auto indent mode for the current window.

set_em_tab_dist(*em-tab-distance*)

Set the emulated tab size. An *em-tab-distance* value of 0 or -1 translates to no emulated tabs. *Em-tab-distance* must be smaller than 1000.

set_fonts(*font-name*, *italic-font-name*, *bold-font-name*, *bold-italic-font-name*)

Set all the fonts used for the current window.

set_highlight_syntax([0 | 1])

Nirvana Editor (NEdit) Help Documentation

Set syntax highlighting mode for the current window. A value of 0 turns it off and a value of 1 turns it on. If no parameters are supplied the option is toggled.

```
set_incremental_backup( [0 | 1] )
```

Set incremental backup mode for the current window. A value of 0 turns it off and a value of 1 turns it on. If no parameters are supplied the option is toggled.

```
set_incremental_search_line( [0 | 1] )
```

Show or hide the incremental search line for the current window. A value of 0 turns it off and a value of 1 turns it on. If no parameters are supplied the option is toggled.

```
set_language_mode( language-mode )
```

Set the language mode for the current window. If the language mode is "" or unrecognized, it will be set to Plain.

```
set_locked( [0 | 1] )
```

This only affects the locked status of a file, not it's read-only status. Permissions are NOT changed. A value of 0 turns it off and a value of 1 turns it on. If no parameters are supplied the option is toggled.

```
set_make_backup_copy( [0 | 1] )
```

Set whether backup copies are made during saves for the current window. A value of 0 turns it off and a value of 1 turns it on. If no parameters are supplied the option is toggled.

```
set_overtypemode( [0 | 1] )
```

Set overtypemode for the current window. A value of 0 turns it off and a value of 1 turns it on. If no parameters are supplied the option is toggled.

```
set_show_line_numbers( [0 | 1] )
```

Show or hide line numbers for the current window. A value of 0 turns it off and a value of 1 turns it on. If no parameters are supplied the option is toggled.

```
set_show_matching( "off" | "delimiter" | "range" )
```

Set show matching (...) mode for the current window.

```
set_match_syntax_based( [0 | 1] )
```

Set whether matching should be syntax based for the current window.

```
set_statistics_line( [0 | 1] )
```

Show or hide the statistics line for the current window. A value of 0 turns it off and a value of 1 turns it on. If no parameters are supplied the option is toggled.

```
set_tab_dist( tab-distance )
```


Set the size of hardware tab spacing. Tab-distance must be a value greater than 0 and no greater than 20.

```
set_use_tabs( [ 0 | 1 ] )
```

Set whether tabs are used for the current window. A value of 0 turns it off and a value of 1 turns it on. If no parameters are supplied the option is toggled.

```
set_wrap_margin( wrap-width )
```

Set the wrap width for text wrapping of the current window. A value of 0 means to wrap at window width.

```
set_wrap_text( "none" | "auto" | "continuous" )
```

Set wrap text mode for the current window.

Keyboard-Only Actions

In addition to the arguments listed in the call descriptions below, any routine involving cursor movement can take the argument "extend", meaning, adjust the primary selection to the new cursor position. Routines which take the "extend" argument as well as mouse dragging operations for both primary and secondary selections can take the optional keyword "rect", meaning, make the selection rectangular. Any routine that accepts the "scrollbar" argument will move the display but not the cursor or selection. Routines that accept the "nobell" argument will fail silently without beeping, when that argument is supplied.

```
backward_character( ["nobell"] )
```

Moves the cursor one character to the left.

```
backward_paragraph( ["nobell"] )
```

Moves the cursor to the beginning of the paragraph, or if the cursor is already at the beginning of a paragraph, moves the cursor to the beginning of the previous paragraph. Paragraphs are defined as regions of text delimited by one or more blank lines.

```
backward_word( ["nobell"] )
```

Moves the cursor to the beginning of a word, or, if the cursor is already at the beginning of a word, moves the cursor to the beginning of the previous word. Word delimiters are user-settable, and defined by the X resource wordDelimiters.

```
beginning_of_file( ["scrollbar"] )
```

Moves the cursor to the beginning of the file.

```
beginning_of_line( ["absolute"] )
```

Moves the cursor to the beginning of the line. If "absolute" is given, always moves to the absolute beginning of line, regardless of the text wrapping mode.

```
beginning_of_selection()
```

Moves the cursor to the beginning of the selection without disturbing the selection.

`copy_clipboard()`

Copies the current selection to the clipboard.

`copy_primary()`

Copies the primary selection to the cursor.

`copy_to()`

If a secondary selection exists, copies the secondary selection to the cursor. If no secondary selection exists, copies the primary selection to the pointer location.

`copy_to_or_end_drag()`

Completes either a secondary selection operation, or a primary drag. If the user is dragging the mouse to adjust a secondary selection, the selection is copied and either inserted at the cursor location, or, if pending-delete is on and a primary selection exists in the window, replaces the primary selection. If the user is dragging a block of text (primary selection), completes the drag operation and leaves the text at its current location.

`cut_clipboard()`

Deletes the text in the primary selection and places it in the clipboard.

`cut_primary()`

Copies the primary selection to the cursor and deletes it at its original location.

`delete_selection()`

Deletes the contents of the primary selection.

`delete_next_character(["nobell"])`

If a primary selection exists, deletes its contents. Otherwise, deletes the character following the cursor.

`delete_previous_character(["nobell"])`

If a primary selection exists, deletes its contents. Otherwise, deletes the character before the cursor.

`delete_next_word(["nobell"])`

If a primary selection exists, deletes its contents. Otherwise, deletes the word following the cursor.

`delete_previous_word(["nobell"])`

If a primary selection exists, deletes its contents. Otherwise, deletes the word before the cursor.

`delete_to_start_of_line(["nobell", "wrap"])`

If a primary selection exists, deletes its contents. Otherwise, deletes the characters between the cursor and the start of the line. If "wrap" is given, deletes to the previous wrap point or beginning of line, whichever is

closest.

```
delete_to_end_of_line( ["nobell", "absolute"] )
```

If a primary selection exists, deletes its contents. Otherwise, deletes the characters between the cursor and the end of the line. If "absolute" is given, always deletes to the absolute end of line, regardless of the text wrapping mode.

```
deselect_all()
```

De-selects the primary selection.

```
end_of_file( ["scrollbar"] )
```

Moves the cursor to the end of the file.

```
end_of_line( ["absolute"] )
```

Moves the cursor to the end of the line. If "absolute" is given, always moves to the absolute end of line, regardless of the text wrapping mode.

```
end_of_selection()
```

Moves the cursor to the end of the selection without disturbing the selection.

```
exchange( ["nobell"] )
```

Exchange the primary and secondary selections.

```
extend_adjust()
```

Attached mouse-movement events to begin a selection between the cursor and the mouse, or extend the primary selection to the mouse position.

```
extend_end()
```

Completes a primary drag-selection operation.

```
extend_start()
```

Begins a selection between the cursor and the mouse. A drag-selection operation can be started with either `extend_start` or `grab_focus`.

```
focus_pane( [relative-pane] | [positive-index] | [negative-index] )
```

Move the focus to the requested pane. Arguments can be specified in the form of a relative-pane ("first", "last", "next", "previous"), a positive-index (numbers greater than 0, 1 is the same as "first") or a negative-index (numbers less than 0, -1 is the same as "last").

```
forward_character()
```

Moves the cursor one character to the right.

Nirvana Editor (NEdit) Help Documentation

`forward_paragraph(["nobell"])`

Moves the cursor to the beginning of the next paragraph. Paragraphs are defined as regions of text delimited by one or more blank lines.

`forward_word(["tail"] ["nobell"])`

Moves the cursor to the beginning of the next word. Word delimiters are user-settable, and defined by the X resource `wordDelimiters`. If the "tail" argument is supplied the cursor will be moved to the end of the current word or the end of the next word, if the cursor is between words.

`grab_focus()`

Moves the cursor to the mouse pointer location, and prepares for a possible drag-selection operation (bound to `extend_adjust`), or multi-click operation (a further `grab_focus` action). If a second invocation of `grab focus` follows immediately, it selects a whole word, or a third, a whole line.

`insert_string("string")`

If pending delete is on and the cursor is inside the selection, replaces the selection with "string". Otherwise, inserts "string" at the cursor location.

`key_select("direction" [,"nobell"])`

Moves the cursor one character in "direction" ("left", "right", "up", or "down") and extends the selection. Same as `forward/backward-character("extend")`, or `process-up/down("extend")`, for compatibility with previous versions.

`move-destination()`

Moves the cursor to the pointer location without disturbing the selection. (This is an unusual way of working. We left it in for compatibility with previous versions, but if you actually use this capability, please send us some mail, otherwise it is likely to disappear in the future.)

`move_to()`

If a secondary selection exists, deletes the contents of the secondary selection and inserts it at the cursor, or if pending-delete is on and there is a primary selection, replaces the primary selection. If no secondary selection exists, moves the primary selection to the pointer location, deleting it from its original position.

`move_to_or_end_drag()`

Completes either a secondary selection operation, or a primary drag. If the user is dragging the mouse to adjust a secondary selection, the selection is deleted and either inserted at the cursor location, or, if pending-delete is on and a primary selection exists in the window, replaces the primary selection. If the user is dragging a block of text (primary selection), completes the drag operation and deletes the text from its current location.

`newline()`

Inserts a newline character. If Auto Indent is on, lines up the indentation of the cursor with the current line.

`newline_and_indent()`

Nirvana Editor (NEdit) Help Documentation

Inserts a newline character and lines up the indentation of the cursor with the current line, regardless of the setting of Auto Indent.

```
newline_no_indent()
```

Inserts a newline character, without automatic indentation, regardless of the setting of Auto Indent.

```
next_page( ["stutter"] ["column"] ["scrollbar"] ["nobell"] )
```

Moves the cursor and scroll forward one page. The parameter "stutter" moves the cursor to the bottom of the display, unless it is already there, otherwise it will page down. The parameter "column" will maintain the preferred column while moving the cursor.

```
page_left( ["scrollbar"] ["nobell"] )
```

Move the cursor and scroll left one page.

```
page_right( ["scrollbar"] ["nobell"] )
```

Move the cursor and scroll right one page.

```
paste_clipboard()
```

Insert the contents of the clipboard at the cursor, or if pending delete is on, replace the primary selection with the contents of the clipboard.

```
previous_page( ["stutter"] ["column"] ["scrollbar"] ["nobell"] )
```

Moves the cursor and scroll backward one page. The parameter "stutter" moves the cursor to the top of the display, unless it is already there, otherwise it will page up. The parameter "column" will maintain the preferred column while moving the cursor.

```
process_bdrag()
```

Same as secondary_or_drag_start for compatibility with previous versions.

```
process_cancel()
```

Cancels the current extend_adjust, secondary_adjust, or secondary_or_drag_adjust in progress.

```
process_down( ["nobell", "absolute"] )
```

Moves the cursor down one line. If "absolute" is given, always moves to the next line in the text buffer, regardless of wrapping.

```
process_return()
```

Same as newline for compatibility with previous versions.

```
process_shift_down( ["nobell", "absolute"] )
```

Same as process_down("extend") for compatibility with previous versions.

Nirvana Editor (NEdit) Help Documentation

```
process_shift_up( ["nobell", "absolute"] )
```

Same as `process_up("extend")` for compatibility with previous versions.

```
process_tab()
```

If tab emulation is turned on, inserts an emulated tab, otherwise inserts a tab character.

```
process_up( ["nobell", "absolute"] )
```

Moves the cursor up one line. If "absolute" is given, always moves to the previous line in the text buffer, regardless of wrapping.

```
raise_window([relative-window] | [positive-index] | [negative-index])
```

Raise the current focused window to the front if no argument is supplied. Arguments can be specified in the form of a relative-window ("first", "last", "next", "previous"), a positive-index (numbers greater than 0, 1 is the same as "last") or a negative-index (numbers less than 0, -1 is the same as "first").

```
scroll_down(nLines)
```

Scroll the display down (towards the end of the file) by nLines.

```
scroll_left( nPixels )
```

Scroll the display left by nPixels.

```
scroll_right( nPixels )
```

Scroll the display right by nPixels.

```
scroll_up( nLines )
```

Scroll the display up (towards the beginning of the file) by nLines.

```
scroll_to_line( lineNumber )
```

Scroll to position line number lineNumber at the top of the pane. The first line of a file is line 1.

```
secondary_adjust()
```

Attached mouse-movement events to extend the secondary selection to the mouse position.

```
secondary_or_drag_adjust()
```

Attached mouse-movement events to extend the secondary selection, or reposition the primary text being dragged. Takes two optional arguments, "copy", and "overlay". "copy" leaves a copy of the dragged text at the site at which the drag began. "overlay" does the drag in overlay mode, meaning the dragged text is laid on top of the existing text, obscuring and ultimately deleting it when the drag is complete.

```
secondary_or_drag_start()
```

Nirvana Editor (NEdit) Help Documentation

To be attached to a mouse down event. Begins drag selecting a secondary selection, or dragging the contents of the primary selection, depending on whether the mouse is pressed inside of an existing primary selection.

secondary_start()

To be attached to a mouse down event. Begin drag selecting a secondary selection.

select_all()

Select the entire file.

self_insert()

To be attached to a key–press event, inserts the character equivalent of the key pressed.

Customizing

Customizing NEdit

NEdit can be customized many different ways. The most important user-settable options are presented in the Preferences menu, including all options that users might need to change during an editing session. Options set in the Default Settings sub-menu of the Preferences menu can be preserved between sessions by selecting Save Defaults, which writes the changes to the preferences file. See the section titled "[Preferences](#)" for more details.

User defined commands can be added to NEdit's Shell, Macro, and window background menus. Dialogs for creating items in these menus can be found under Customize Menus in the Default Settings sub menu of the Preferences menu.

For users who depend on NEdit every day and want to tune every excruciating detail, there are also X resources for tuning a vast number of such details, down to the color of each individual button. See the section "[X Resources](#)" for more information, as well as a list of selected resources.

The most common reason customizing your X resources for NEdit, however, is key binding. While limited key binding can be done through Preferences settings (Preferences → Default Settings → Customize Menus), you can really only add keys this way, and each key must have a corresponding menu item. Any significant changes to key binding should be made via the Translations resource and menu accelerator resources. The sections titled "[Key Binding](#)" and "[X Resources](#)" have more information.

Preferences

The Preferences menu allows you to set options for both the current editing window, and default values for newly created windows and future NEdit sessions. Options in the Preferences menu itself (not in the Default Settings sub-menu) take effect immediately and refer to the current window only. Options in the Default Settings sub-menu provide initial settings for future windows created using the New or Open commands; options affecting all windows are also set here. Preferences set in the Default Settings sub-menu can be saved in a file that is automatically read by NEdit at startup time, by selecting Save Defaults.

Preferences Menu

Default Settings

Menu of initial settings for future windows. Generally the same as the options in the main part of the menu, but apply as defaults for future windows created during this NEdit session. These settings can be saved using the Save Defaults command below, to be loaded automatically each time NEdit is started.

Save Defaults

Save the default options as set under Default Settings for future NEdit sessions.

Statistics Line

Nirvana Editor (NEdit) Help Documentation

Show the full file name, line number, and length of the file being edited.

Incremental Search Line

Keep the incremental search bar (Search → Find Incremental) permanently displayed at the top of the window.

Show Line Numbers

Display line numbers to the right of the text.

Language Mode

Tells NEdit what language (if any) to assume, for selecting language-specific features such as highlight patterns and smart indent macros, and setting language specific preferences like word delimiters, tab emulation, and auto-indent. See Features for Programming → [Programming with NEdit](#) for more information.

Auto Indent

Setting Auto Indent "on" maintains a running indent (pressing the Return key will line up the cursor with the indent level of the previous line). If smart indent macros are available for the current language mode, smart indent can be selected and NEdit will attempt to guess proper language indentation for each new line. See Help → Features for Programming → Automatic Indent for more information.

Wrap

Choose between two styles of automatic wrapping or none. Auto Newline wrap, wraps text at word boundaries when the cursor reaches the right margin, by replacing the space or tab at the last word boundary with a newline character. Continuous Wrap wraps long lines which extend past the right margin. Continuous Wrap mode is typically used to produce files where newlines are omitted within paragraphs, to make text filling automatic (a kind of poor-man's word processor). Text of this style is common on Macs and PCs but is not necessarily supported very well under Unix (except in programs which deal with e-mail, for which it is often the format of choice).

Wrap Margin

Set margin for Auto Newline Wrap, Continuous Wrap, and Fill Paragraph. Lines may, be wrapped at the right margin of the window, or the margin can be set at a specific column.

Tabs

Set the tab distance (number of characters between tab stops) for tab characters, and control tab emulation and use of tab characters in padding and emulated tabs.

Text Font...

Change the font(s) used to display text (fonts for menus and dialogs must be set using X resources for the text area of the window). See below for more information.

Colors...

Nirvana Editor (NEdit) Help Documentation

Change the colors used to display text. The "Matching (..)" fields change the colors that matching parens, brackets and braces are flashed when the "Show Matching (..)" option is enabled. Note that the foreground colors for plain text, selected text, and matching paren flashing only apply when syntax highlighting is disabled. When syntax highlighting is enabled, text (even text that appears plain) will always be colored according to its highlighting style. (For information on changing syntax highlighting styles and matching patterns use see Help -> Features for Programming -> [Syntax Highlighting](#).)

Highlight Syntax

If NEdit recognizes the language being edited, and highlighting patterns are available for that language, use fonts and colors to enhance viewing of the file. (See Help -> Features for Programming -> Syntax Highlighting for more information.

Make Backup Copy

On Save, write a backup copy of the file as it existed before the Save command with the extension .bck (Unix only).

Incremental Backup

Periodically make a backup copy of the file being edited under the name `~filename` on Unix or `_filename` on VMS (see [Crash Recovery](#)).

Show Matching (..)

Momentarily highlight matching parenthesis, brackets, and braces, or the range between them, when one of these characters is typed, or when the insertion cursor is positioned after it. Delimiter only highlights the matching delimiter, while Range highlights the whole range of text between the matching delimiters.

Optionally, the matching can make use of syntax information if syntax highlighting is enabled. Alternatively, the matching is purely character based. In general, syntax based matching results in fewer false matches.

Overtyping

In overtype mode, new characters entered replace the characters in front of the insertion cursor, rather than being inserted before them.

Read Only

Lock the file against accidental modification. This temporarily prevents the file from being modified in this NEdit session. Note that this is different from setting the file protection.

Preferences -> Default Settings Menu

Options in the Preferences -> Default Settings menu have the same meaning as those in the top-level Preferences menu, except that they apply to future NEdit windows and future NEdit sessions if saved with the Save Defaults command. Additional options which appear in this menu are:

Language Modes

Define language recognition information (for determining language mode from file name or content) and set language specific preferences.

Nirvana Editor (NEdit) Help Documentation

Tag Collisions

How to react to multiple tags for the same name. Tags are described in the section: Features for Programmers -> Finding Declarations (ctags). In Show All mode, all matching tags are displayed in a dialog. In Smart mode, if one of the matching tags is in the current window, that tag is chosen, without displaying the dialog.

Customize Menus

Add/remove items from the Shell, Macro, and window background menus (see below).

Customize Window Title

Opens a dialog where the information to be displayed in the window's title field can be defined and tested. The dialog contains a Help button, providing further information about the options available.

Searching

Options for controlling the behavior of Find and Replace commands:

Verbose – Presents search results in dialog form, asks before wrapping a search back around the beginning (or end) of the file (unless Beep On Search Wrap is turned on).

Wrap Around – Search and Replace operations wrap around the beginning (or end) of the file.

Beep On Search Wrap – Beep when Search and Replace operations wrap around the beginning (or end) of the file (only if Wrap Around is turned on).

Keep Dialogs Up – Don't pop down Replace and Find boxes after searching.

Default Search Style – Initial setting for search type in Find and Replace dialogs.

Default Replace Scope – [THIS OPTION IS ONLY PRESENT WHEN NEDIT WAS COMPILED WITH THE

`-DREPLACE_SCOPE` FLAG TO SELECT AN ALTERNATIVE REPLACE DIALOG LAYOUT.]

Initial setting for the scope in the Replace/Find dialog, when a selection exists. It can be either "In Window", "In Selection", or "Smart". "Smart" results in "In Window" if the size of the selection is smaller than 1 line, and to "In Selection" otherwise.

Syntax Highlighting

Program and configure enhanced text display for new or supported languages (See Features for Programming -> [Syntax Highlighting](#)).

Terminate with Line Break on Save

Some UNIX tools expect that files end with a line feed. If this option is activated, NEdit will append one if required.

Sort Open Prev. Menu

Nirvana Editor (NEdit) Help Documentation

Option to order the File → Open Previous menu alphabetically, versus in order of last access.

Popups Under Pointer

Display pop-up dialogs centered on the current mouse position, as opposed to centered on the parent window. This generally speeds interaction, and is essential for users who set their window managers so keyboard focus follows the mouse.

Warnings

Options for controlling the popping up of warning dialogs:

File Modified Externally – Pop up a warning dialog when files get changed external to NEdit.

Check Modified File Contents – If external file modification warnings are requested, also check the file contents iso. only the modification date.

Exit Warnings – Ask before exiting when two or more files are open in an NEdit session.

Initial Window Size

Default size for new windows.

Changing Font(s)

The font used to display text in NEdit is set under Preferences → Text Font (for the current window), or Preferences → Default Settings Text Font (for future windows). These dialogs also allow you to set fonts for syntax highlighting. If you don't intend to use syntax highlighting, you can ignore most of the dialog, and just set the field labeled Primary Font.

Unless you are absolutely certain about the types of files that you will be editing with NEdit, you should choose a fixed-spacing font. Many, if not most, plain-text files are written expecting to be viewed with fixed character spacing, and will look wrong with proportional spacing. NEdit's filling, wrapping, and rectangular operations will also work strangely if you choose a proportional font.

Note that in the font browser (the dialog brought up by the Browse... button), the subset of fonts which are shown is narrowed depending on the characteristics already selected. It is therefore important to know that you can unselect characteristics from the lists by clicking on the selected items a second time.

Fonts for syntax highlighting should ideally match the primary font in both height and spacing. A mismatch in spacing will result in similar distortions as choosing a proportional font: column alignment will sometimes look wrong, and rectangular operations, wrapping, and filling will behave strangely. A mismatch in height will cause windows to re-size themselves slightly when syntax highlighting is turned on or off, and increase the inter-line spacing of the text. Unfortunately, on some systems it is hard to find sets of fonts which match exactly in height.

Customizing Menus

You can add or change items in the Shell, Macro, and window background menus under Preferences → Default Settings → Customize Menu. When you choose one of these, you will see a dialog with a list of the current user-configurable items from the menu on the left. To change an existing item, select it from the list,

Nirvana Editor (NEdit) Help Documentation

and its properties will appear in the remaining fields of the dialog, where you may change them. Selecting the item "New" from the list allows you to enter new items in the menu.

Hopefully most of the characteristics are self explanatory, but here are a few things to note:

Accelerator keys are keyboard shortcuts which appear on the right hand side of the menus, and allow you avoid pulling down the menu and activate the command with a single keystroke. Enter accelerators by typing the keys exactly as you would to activate the command.

Mnemonics are a single letter which should be part of the menu item name, which allow users to traverse and activate menu items by typing keys when the menu is pulled down.

In the Shell Command field of the Shell Commands dialog, the % character expands to the name (including directory path) of the file in the window. To include a % character in the command, use %%.

The Menu Entry field can contain special characters for constructing hierarchical sub-menus, and for making items which appear only in certain language modes. The right angle bracket character ">" creates a sub-menu. The name of the item itself should be the last element of the path formed from successive sub-menu names joined with ">". Menu panes are called in to existence simply by naming them as part of a Menu Entry name. To put several items in the same sub-menu, repeat the same hierarchical sequence for each. For example, in the Macro Commands dialog, two items with menu entries: a>b>c and a>b>d would create a single sub menu under the macro menu called "a", which would contain a single sub-menu, b, holding the actual items, c and d:

```
+-----+
| a > | | b > | | c |
+-----+ | d |
          +-----+
```

To qualify a menu entry with a language mode, simply add an at-sign "@" at the end of the menu command, followed (no space) by a language mode name. To make a menu item which appears in several language modes, append additional @s and language mode names. For example, an item with the menu entry:

```
Make C Prototypes@C@C++
```

would appear only in C and C++ language modes, and:

```
Make Class Template@C++
```

would appear only in C++ mode.

Menu items with no qualification appear in all language modes.

If a menu item is followed by the single language qualification "@*", that item will appear only if there are no applicable language-specific items of the same name in the same submenu. For example, if you have the following three entries in the same menu:

```
Make Prototypes@C@C++
Make Prototypes@Java
Make Prototypes@*
```

The first will be available when the language mode is C or C++, the second when the language mode is Java,

and for all other language modes (including the "Plain" non-language mode). If the entry:

```
Make Prototypes
```

also exists, this will always appear, meaning that the menu will always have two "Make Prototypes" entries, whatever the language mode.

The NEdit Preferences File

The NEdit saved preferences file is an X resource file, and its contents can be moved into another X resource file (see [X Resources](#)). One reason for doing so would be to attach server specific preferences, such as a default font to a particular X server. Another reason for moving preferences into the X resource file would be to keep preferences menu options and resource settable options together in one place. Though the files are the same format, additional resources should not be added to the preference file since NEdit modifies this file by overwriting it completely. Note also that the contents of the preference file take precedence over the values of X resources. Using Save Defaults after moving the contents of your preference file to your .Xdefaults file will re-create the preference file, interfering with the options that you have moved. The location of NEdit's preferences file depends on your environment:

- The default place for the file is '\$HOME/.nedit/nedit.rc',
- if the variable \$NEDIT_HOME is set in your environment it is located at '\$NEDIT_HOME/nedit.rc',
- you may also use old-style run control files; in this case, the preferences are stored in \$HOME/.nedit.

(For VMS, the file is in '\$NEDIT_HOME/nedit.rc' if \$NEDIT_HOME is set, in '\$SYS\$LOGIN:.nedit' otherwise.)

Sharing Customizations with Other NEdit Users

If you have written macro or shell menu commands, highlight patterns, or smart-indent macros that you want to share with other NEdit users, you can make a file which they can load into their NEdit environment.

To load such a file, start NEdit with the command:

```
nedit -import <file>
```

In the new NEdit session, verify that the imported patterns or macros do what you want, then select Preferences → Save Defaults. Saving incorporates the changes into the nedit preferences file, so the next time you run NEdit, you will not have to import the distribution file.

Loading a customization file is automated, but creating one is not. To produce a file to be imported by other users, you must make a copy of your own NEdit configuration file, and edit it, by hand, to remove everything but the few items of interest to the recipient. Leave only the individual resource(s), and within those resources, only the particular macro, pattern, style, etc, that you wish to exchange.

For example, to share a highlighting pattern set, you would include the patterns, any new styles you added, and language mode information only if the patterns are intended to support a new language rather than updating an existing one. For example:

```
nedit.highlightPatterns:\
  My Language:1:0{\n\
    Comment:"#":"$":Comment::\n\
```

Nirvana Editor (NEdit) Help Documentation

```
        Loop Header:"^[ \\t]*loop:":::Loop::\n\  
    }  
nedit.languageModes: My Language:.my:::::  
nedit.styles: Loop:blue:Bold
```

Resources are in the format of X resource files, but the format of text within multiple-item resources like highlight patterns, language modes, macros, styles, etc., are private to NEdit. Each resource is a string which ends at the first newline character not escaped with \, so you must be careful about how you treat ends of lines. While you can generally just cut and paste indented sections, if something which was originally in the middle of a resource string is now at the end, you must remove the \ line continuation character(s) so it will not join the next line into the resource. Conversely, if something which was originally at the end of a resource is now in the middle, you'll have to add continuation character(s) to make sure that the resource string is properly continued from beginning to end, and possibly newline character(s) (\n) to make sure that it is properly separated from the next item.

X Resources

NEdit has additional options to those provided in the Preferences menu which are set using X resources. Like most other X programs, NEdit can be customized to vastly unnecessary proportions, from initial window positions down to the font and shadow colors of each individual button (A complete discussion of how to do this is left to books on the X Window System). Key binding (see "[Key Binding](#)") is one of the most useful of these resource settable options.

X resources are usually specified in a file called .Xdefaults or .Xresources in your home directory (on VMS this is sys\$login:decw\$xdefaults.dat). On some systems, this file is read and its information attached to the X server (your screen) when you start X. On other systems, the .Xdefaults file is read each time you run an X program. When X resource values are attached to the X server, changes to the resource file are not available to application programs until you either run the xrb program with the appropriate file as input, or re-start the X server.

Selected X Resource Names

The following are selected NEdit resource names and default values for NEdit options not settable via the Preferences menu (for preference resource names, see your NEdit preference file):

nedit.tagFile: (not defined)

This can be the name of a file, or multiple files separated by a colon (:) character, of the type produced by Exuberant Ctags or the Unix ctags command, which NEdit will load at startup time (see [ctag support](#)). The tag file provides a database from which NEdit can automatically open files containing the definition of a particular subroutine or data type.

nedit.alwaysCheckRelativeTagsSpecs: True

When this resource is set to True, and there are tag files specified (with the nedit.tagFile resource, see above) as relative paths, NEdit will evaluate these tag value paths whenever a file is opened. All accessible tag files will be loaded at this time. When this resource value is False, relative path tag specifications will only be evaluated at NEdit startup time.

nedit.shell: /bin/csh

Nirvana Editor (NEdit) Help Documentation

(Unix systems only) The Unix shell (command interpreter) to use for executing commands from the Shell menu

nedit.wordDelimiters: .,/\\`'!@#%^&*()-=+{ } [] " : ; < > ?

The characters, in addition to blanks and tabs, which mark the boundaries between words for the move-by-word (Ctrl+Arrow) and select-word (double click) commands. Note that this default value may be overridden by the setting in Preferences → Default Settings → Language Modes....

nedit.remapDeleteKey: False

Setting this resource to True forcibly maps the delete key to backspace. This can be helpful on systems where the bindings have become tangled, and in environments which mix systems with PC style keyboards and systems with DEC and Macintosh keyboards. Theoretically, these bindings should be made using the standard X/Motif mechanisms, outside of NEdit. In practice, some environments where users access several different systems remotely, can be very hard to configure. If you've given up and are using a backspace key halfway off the keyboard because you can't figure out the bindings, set this to True.

nedit.typingHidesPointer: False

Setting this resource to True causes the mouse pointer to be hidden when you type in the text area. As soon as the mouse pointer is moved, it will reappear. This is useful to stop the mouse pointer from obscuring text.

nedit.overrideDefaultVirtualKeyBindings: Auto

Motif uses a virtual key binding mechanism that shares the bindings between different Motif applications. When a first Motif application is started, it installs some default virtual key bindings and any other Motif application that runs afterwards, simply reuses them. Obviously, if the first application installs an invalid set, all others applications may have problems.

In the past, NEdit has been the victim of invalid bindings installed by other applications several times. Through this resource, NEdit can be instructed to ignore the bindings installed by other applications, and use its own private bindings. By default, NEdit tries to detect invalid bindings and ignore them automatically (Auto). Optionally, NEdit can be told to always keep the installed bindings (Never), or to always override them (Always).

nedit.stdOpenDialog: False

Setting this resource to True restores the standard Motif style of Open dialog. NEdit file open dialogs are missing a text field at the bottom of the dialog, where the file name can be entered as a string. The field is removed in NEdit to encourage users to type file names in the list, a non-standard, but much faster method for finding files.

nedit.bgMenuButton: ~Shift~Ctrl~Meta~Alt<Btn3Down>

Specification for mouse button / key combination to post the background menu (in the form of an X translation table event specification). The event specification should be as specific as possible, since it will override less specific translation table entries.

nedit.maxPrevOpenFiles: 30

Nirvana Editor (NEdit) Help Documentation

Number of files listed in the Open Previous sub-menu of the File menu. Setting this to zero disables the Open Previous menu item and maintenance of the NEdit file history file.

nedit.printCommand: (system specific)

Command used by the print dialog to print a file, such as, lp, lpr, etc.. The command must be capable of accepting input via stdin (standard input).

nedit.printCopiesOption: (system specific)

Option name used to specify multiple copies to the print command. If the option should be separated from its argument by a space, leave a trailing space. If blank, no "Number of Copies" item will appear in the print dialog.

nedit.printQueueOption: (system specific)

Option name used to specify a print queue to the print command. If the option should be separated from its argument by a space, leave a trailing space. If blank, no "Queue" item will appear in the print dialog.

nedit.printNameOption: (system specific)

Option name used to specify a job name to the print command. If the option should be separated from its argument by a space, leave a trailing space. If blank, no job or file name will be attached to the print job or banner page.

nedit.printHostOption: (system specific)

Option name used to specify a host name to the print command. If the option should be separated from its argument by a space, leave a trailing space. If blank, no "Host" item will appear in the print dialog.

nedit.printDefaultQueue: (system specific)

The name of the default print queue. Used only to display in the print dialog, and has no effect on printing.

nedit.visualID: Best

If your screen supports multiple visuals (color mapping models), this resource allows you to manually choose among them. The default value of "Best" chooses the deepest (most colors) visual available. Since NEdit does not depend on the specific characteristics of any given color model, Best probably IS the best choice for everyone, and the only reason for setting this resource would be to patch around some kind of X server problem. The resource may also be set to "Default", which chooses the screen's default visual (often a color-mapped, PseudoColor, visual for compatibility with older X applications). It may also be set to a numeric visual-id value (use xdpyinfo to see the list of visuals supported by your display), or a visual class name: PseudoColor, DirectColor, TrueColor, etc..

If you are running under a themed environment (like KDE or CDE) that places its colors in a shallow visual, and you'd rather have that color scheme instead of more colors available, then you may need set the visual to "Default" so that NEdit doesn't choose one with more colors. (The reason for this is: if the "best" visual is not the server's default, then NEdit cannot use the colors provided by your environment. NEdit will fall back to its own default color scheme.)

nedit.installColormap: False

Nirvana Editor (NEdit) Help Documentation

Force the installation of a private colormap. If you have a humble 8-bit color display, and netscape is hogging all of the color cells, you may want to try turning this on. On most systems, this will result in colors flashing wildly when you switch between NEdit and other applications. But a few systems (SGI) have hardware support for multiple simultaneous colormaps, and applications with installed colormaps are well behaved.

`nedit.findReplaceUsesSelection: False`

Controls if the Find and Replace dialogs are automatically loaded with the contents of the primary selection.

`nedit.stickyCaseSenseButton: True`

Controls if the "Case Sensitive" buttons in the Find and Replace dialogs and the incremental search bar maintain a separate state for literal and regular expression searches. Moreover, when set to True, by default literal searches are case insensitive and regular expression searches are case sensitive. When set to False, the "Case Sensitive" buttons are independent of the "Regular Expression" toggle.

`nedit.printDefaultHost: (system specific)`

The node name of the default print host. Used only to display in the print dialog, and has no effect on printing.

`nedit.multiClickTime: (system specific)`

Maximum time in milliseconds allowed between mouse clicks within double and triple click actions.

`nedit*scrollBarPlacement: BOTTOM_LEFT`

How scroll bars are placed in NEdit windows, as well as various lists and text fields in the program. Other choices are: `BOTTOM_RIGHT`, `TOP_LEFT`, or `TOP_RIGHT`.

`nedit*text.autoWrapPastedText: False`

When Auto Newline Wrap is turned on, apply automatic wrapping (which normally only applies to typed text) to pasted text as well.

`nedit*text.heavyCursor: False`

For monitors with poor resolution or users who have difficulty seeing the cursor, makes the cursor in the text editing area of the window heavier and darker.

`nedit*text.cursorVPadding: 0`

Number of lines to keep the cursor away from the top or bottom line of the window. Keyboard operations that would cause the cursor to get closer than this distance cause the window to scroll up or down instead, except at the beginning of the file. Mouse operations are not affected.

`nedit*text.blinkRate: 500`

Blink rate of the text insertion cursor in milliseconds. Set to zero to stop blinking.

`nedit*text.Translations:`

Modifies key bindings (see below).

Nirvana Editor (NEdit) Help Documentation

`nedit*foreground: black`

Default foreground color for menus, dialogs, scroll bars, etc..

`nedit*background: #b3b3b3`

Default background color for menus, dialogs, scroll bars, etc..

`nedit*calltipForeground: black`

Foreground color for calltips

`nedit*calltipBackground: LemonChiffon1`

Background color for calltips

`nedit*fontList: helvetica medium 12 points`

Default font for menus, dialogs, scroll bars, etc..

`nedit.helpFont: helvetica medium 12 points`

Font used for displaying online help.

`nedit.boldHelpFont: helvetica bold 12 points`

Bold font for online help.

`nedit.italicHelpFont: helvetica italic 12 points`

Italic font for online help.

`nedit.fixedHelpFont: courier medium 12 points`

Fixed font for online help.

`nedit.boldFixedHelpFont: courier bold 12 points`

Fixed bold for online help.

`nedit.italicFixedHelpFont: courier italic 12 points`

Fixed italic font for online help.

`nedit.h1HelpFont: helvetica bold 14 points`

Font for level-1 titles in help text.

`nedit.h2HelpFont: helvetica bold italic 12 points`

Font for level-2 titles in help text.

`nedit.h3HelpFont: courier bold 12 points`

Nirvana Editor (NEdit) Help Documentation

Font for level-3 titles in help text.

```
nedit.helpLinkFont: helvetica medium 12 points
```

Font for hyperlinks in the help text

```
nedit.helpLinkColor: #009900
```

Color for hyperlinks in the help text

```
nedit.backlightCharTypes: 0-8,10-31,127:red;9:#dedede;32-126,160-255:#f0f0f0;128-159:orange
```

NOTE: backlighting is *experimental* (see "[Programming with NEdit](#)").

A string specifying character classes as ranges of ASCII values followed by the color to be used as their background colors. The format is:

```
low[-high]{,low[-high]}:color{;low-high{,low[-high]}:color}
```

where low and high are ASCII values.

For example:

```
32-255:#f0f0f0;1-31,127:red;128-159:orange;9-13:#e5e5e5
```

```
nc.autoStart: True
```

Whether the nc program should automatically start an NEdit server (without prompting the user) if an appropriate server is not found.

```
nc.serverCommand: nedit -server
```

Command used by the nc program to start an NEdit server.

```
nc.timeOut: 10
```

Basic time-out period used in communication with an NEdit server (seconds).

The following are Selected widget names (to which you may append .background, .foreground, .fontList, etc., to change colors, fonts and other characteristics):

```
nedit*statsAreaForm
```

Statistics line and incremental search bar. To get consistent results across the entire stats line and the incremental search bar, use '*' rather than '.' to separate the resource name. For example, to set the foreground color of both components use:

```
nedit*statsAreaForm*foreground
```

instead of:

```
nedit*statsAreaForm.foreground
```

`nedit*menuBar`

Top-of-window menu-bar.

`nedit*textHorScrollBar`

Horizontal scroll bar.

`nedit*textVertScrollBar`

Vertical scroll bar.

Key Binding

There are several ways to change key bindings in NEdit. The easiest way to add a new key binding in NEdit is to define a macro in Preferences → Default Settings → Customize Menus → Macro Menu. However, if you want to change existing bindings or add a significant number of new key bindings you will need to do so via X resources.

Before reading this section, you must understand how to set X resources (see the help section "[X Resources](#)"). Since setting X resources is tricky, it is also helpful when working on key-binding, to set some easier-to-verify resource at the same time, as a simple check that the NEdit program is actually seeing your changes. The appres program is also very helpful in checking that the resource settings that you make, actually reach the program for which they are intended in the correct form.

Key Binding in General

Keyboard commands are associated with editor action routines through two separate mechanisms in NEdit. Commands which appear in pull-down menus have individual resources designating a keyboard equivalent to the menu command, called an accelerator key. Commands which do not have an associated menu item are bound to keys via the X toolkit translation mechanism. The methods for changing these two kinds of bindings are quite different.

Key Binding Via Translations

The most general way to bind actions to keys in NEdit is to use the translation table associated with the text widget. To add a binding to Alt+Y to insert the string "Hi!", for example, add lines similar to the following to your X resource file:

```
NEdit*text.Translations: #override \n\  
Alt<Key>y: insert_string("Hi!") \n
```

The Help topic "[Action Routines](#)" lists the actions available to be bound.

Translation tables map key and mouse presses, window operations, and other kinds of events, to actions. The syntax for translation tables is simplified here, so you may need to refer to a book on the X window system for more detailed information.

Note that accelerator resources (discussed below) override translations, and that most Ctrl+letter and

Alt+letter combinations are already bound to an accelerator key. To use one of these combinations from a translation table, therefore, you must first un-bind the original menu accelerator.

A resource for changing a translation table consists of a keyword; #override, #augment, or #replace; followed by lines (separated by newline characters) pairing events with actions. Events begin with modifiers, like Ctrl, Shift, or Alt, followed by the event type in <>. BtnDown, Btn1Down, Btn2Down, Btn1Up, Key, KeyUp are valid event types. For key presses, the event type is followed by the name of the key. You can specify a combination of events, such as a sequence of key presses, by separating them with commas. The other half of the event/action pair is a set of actions. These are separated from the event specification by a colon and from each other by spaces. Actions are names followed by parentheses, optionally containing one or more parameters separated by commas.

Changing Menu Accelerator Keys

The menu shortcut keys shown at the right of NEdit menu items can also be changed via X resources. Each menu item has two resources associated with it, accelerator, the event to trigger the menu item; and acceleratorText, the string shown in the menu. The form of the accelerator resource is the same as events for translation table entries discussed above, though multiple keys and other subtleties are not allowed. The resource name for a menu is the title in lower case, followed by "Menu", the resource name of menu item is the name in lower case, run together, with words separated by caps, and all punctuation removed. For example, to change Cut to Ctrl+X, you would add the following to your .Xdefaults file:

```
nedit*editMenu.cut.accelerator: Ctrl<Key>x  
nedit*editMenu.cut.acceleratorText: Ctrl+X
```

Accelerator keys with optional shift key modifiers, like Find..., have an additional accelerator resource with Shift appended to the name. For example:

```
nedit*searchMenu.find.acceleratorText: [Shift]Alt+F  
nedit*searchMenu.find.accelerator: Alt<Key>f  
nedit*searchMenu.findShift.accelerator: Shift Alt<Key>f
```

Highlighting Patterns

Writing Syntax Highlighting Patterns

Patterns are the mechanism by which language syntax highlighting is implemented in NEdit (see [Syntax Highlighting](#) under the heading of Features for Programming). To create syntax highlighting patterns for a new language, or to modify existing patterns, select "Recognition Patterns" from "Syntax Highlighting" sub-section of the "Default Settings" sub-menu of the "Preferences" menu.

First, a word of caution. As with regular expression matching in general, it is quite possible to write patterns which are so inefficient that they essentially lock up the editor as they recursively re-examine the entire contents of the file thousands of times. With the multiplicity of patterns, the possibility of a lock-up is significantly increased in syntax highlighting. When working on highlighting patterns, be sure to save your work frequently.

NEdit's syntax highlighting is unusual in that it works in real-time (as you type), and yet is completely programmable using standard regular expression notation. Other syntax highlighting editors usually fall either into the category of fully programmable but unable to keep up in real-time, or real-time but limited

programmability. The additional burden that NEdit places on pattern writers in order to achieve this speed/flexibility mix, is to force them to state self-imposed limitations on the amount of context that patterns may examine when re-parsing after a change. While the "Pattern Context Requirements" heading is near the end of this section, it is not optional, and must be understood before making any any serious effort at pattern writing.

In its simplest form, a highlight pattern consists of a regular expression to match, along with a style representing the font and color for displaying any text which matches that expression. To bold the word, "highlight", wherever it appears the text, the regular expression simply would be the word "highlight". The style (selected from the menu under the heading of "Highlight Style") determines how the text will be drawn. To bold the text, either select an existing style, such as "Keyword", which bolds text, or create a new style and select it under Highlight Style.

The full range of regular expression capabilities can be applied in such a pattern, with the single caveat that the expression must conclusively match or not match, within the pre-defined context distance (as discussed below under Pattern Context Requirements).

To match longer ranges of text, particularly any constructs which exceed the requested context, you must use a pattern which highlights text between a starting and ending regular expression match. To do so, select "Highlight text between starting and ending REs" under "Matching", and enter both a starting and ending regular expression. For example, to highlight everything between double quotes, you would enter a double quote character in both the starting and ending regular expression fields. Patterns with both a beginning and ending expression span all characters between the two expressions, including newlines.

Again, the limitation for automatic parsing to operate properly is that both expressions must match within the context distance stated for the pattern set.

With the ability to span large distances, comes the responsibility to recover when things go wrong. Remember that syntax highlighting is called upon to parse incorrect or incomplete syntax as often as correct syntax. To stop a pattern short of matching its end expression, you can specify an error expression, which stops the pattern from gobbling up more than it should. For example, if the text between double quotes shouldn't contain newlines, the error expression might be "\$". As with both starting and ending expressions, error expressions must also match within the requested context distance.

Coloring Sub-Expressions

It is also possible to color areas of text within a regular expression match. A pattern of this type associates a style with sub-expressions references of the parent pattern (as used in regular expression substitution patterns, see the NEdit Help menu item on [Regular Expressions](#)). Sub-expressions of both the starting and ending patterns may be colored. For example, if the parent pattern has a starting expression "\<", and end expression "\>", (for highlighting all of the text contained within angle brackets), a sub-pattern using "&" in both the starting and ending expression fields could color the brackets differently from the intervening text. A quick shortcut to typing in pattern names in the Parent Pattern field is to use the middle mouse button to drag them from the Patterns list.

Hierarchical Patterns

A hierarchical sub-pattern, is identical to a top level pattern, but is invoked only between the beginning and ending expression matches of its parent pattern. Like the sub-expression coloring patterns discussed above, it is associated with a parent pattern using the Parent Pattern field in the pattern specification. Pattern names can be dragged from the pattern list with the middle mouse button to the Parent Pattern field.

After the start expression of the parent pattern matches, the syntax highlighting parser searches for either the parent's end pattern or a matching sub-pattern. When a sub-pattern matches, control is not returned to the parent pattern until the entire sub-pattern has been parsed, regardless of whether the parent's end pattern appears in the text matched by the sub-pattern.

The most common use for this capability is for coloring sub-structure of language constructs (smaller patterns embedded in larger patterns). Hierarchical patterns can also simplify parsing by having sub-patterns "hide" special syntax from parent patterns, such as special escape sequences or internal comments.

There is no depth limit in nesting hierarchical sub-patterns, but beyond the third level of nesting, automatic re-parsing will sometimes have to re-parse more than the requested context distance to guarantee a correct parse (which can slow down the maximum rate at which the user can type if large sections of text are matched only by deeply nested patterns).

While this is obviously not a complete hierarchical language parser it is still useful in many text coloring situations. As a pattern writer, your goal is not to completely cover the language syntax, but to generate colorings that are useful to the programmer. Simpler patterns are usually more efficient and also more robust when applied to incorrect code.

Deferred (Pass-2) Parsing

NEdit does pattern matching for syntax highlighting in two passes. The first pass is applied to the entire file when syntax highlighting is first turned on, and to new ranges of text when they are initially read or pasted in. The second pass is applied only as needed when text is exposed (scrolled in to view).

If you have a particularly complex set of patterns, and parsing is beginning to add a noticeable delay to opening files or operations which change large regions of text, you can defer some of that parsing from startup time, to when it is actually needed for viewing the text. Deferred parsing can only be used with single expression patterns, or begin/end patterns which match entirely within the requested context distance. To defer the parsing of a pattern to when the text is exposed, click on the Pass-2 pattern type button in the highlight patterns dialog.

Sometimes a pattern can't be deferred, not because of context requirements, but because it must run concurrently with pass-1 (non-deferred) patterns. If they didn't run concurrently, a pass-1 pattern might incorrectly match some of the characters which would normally be hidden inside of a sequence matched by the deferred pattern. For example, C has character constants enclosed in single quotes. These typically do not cross line boundaries, meaning they can be parsed entirely within the context distance of the C pattern set and should be good candidates for deferred parsing. However, they can't be deferred because they can contain sequences of characters which can trigger pass-one patterns. Specifically, the sequence, "\", contains a double quote character, which would be matched by the string pattern and interpreted as introducing a string.

Pattern Context Requirements

The context requirements of a pattern set state how much additional text around any change must be examined to guarantee that the patterns will match what they are intended to match. Context requirements are a promise by NEdit to the pattern writer, that the regular expressions in his/her patterns will be matched against at least <line context> lines and <character context> characters, around any modified text. Combining line and character requirements guarantee that both will be met.

Automatic re-parsing happens on EVERY KEYSTROKE, so the amount of context which must be examined is very critical to typing efficiency. The more complicated your patterns, the more critical the context

becomes. To cover all of the keywords in a typical language, without affecting the maximum rate at which users can enter text, you may be limited to just a few lines and/or a few hundred characters of context.

The default context distance is 1 line, with no minimum character requirement. There are several benefits to sticking with this default. One is simply that it is easy to understand and to comply with. Regular expression notation is designed around single line matching. To span lines in a regular expression, you must explicitly mention the newline character "\n", and matches which are restricted to a single line are virtually immune to lock-ups. Also, if you can code your patterns to work within a single line of context, without an additional character-range context requirement, the parser can take advantage the fact that patterns don't cross line boundaries, and nearly double its efficiency over a one-line and 1-character context requirement. (In a single line context, you are allowed to match newlines, but only as the first and/or last character.)

Smart Indent Macros

Smart indent macros can be written for any language, but are usually more difficult to write than highlighting patterns. A good place to start, of course, is to look at the existing macros for C and C++.

Smart indent macros for a language mode consist of standard NEdit macro language code attached to any or all of the following three activation conditions: 1) When smart indent is first turned on for a text window containing code of the language, 2) When a newline is typed and smart indent is expected, 3) after any character is typed. To attach macro code to any of these code "hooks", enter it in the appropriate section in the Preferences -> Default Settings -> Auto Indent -> Program Smart Indent dialog.

Typically most of the code should go in the initialization section, because that is the appropriate place for subroutine definitions, and smart indent macros are complicated enough that you are not likely to want to write them as one monolithic run of code. You may also put code in the Common/Shared Initialization section (accessible through the button in the upper left corner of the dialog). Unfortunately, since the C/C++ macros also reside in the common/shared section, when you add code there, you run some risk of missing out on future upgrades to these macros, because your changes will override the built-in defaults.

The newline macro is invoked after the user types a newline, but before the newline is entered in the buffer. It takes a single argument (\$1) which is the position at which the newline will be inserted. It must return the number of characters of indentation the line should have, or -1. A return value of -1 means to do a standard auto-indent. You must supply a newline macro, but the code: "return -1" (auto-indent), or "return 0" (no indent) is sufficient.

The type-in macro takes two arguments. \$1 is the insert position, and \$2 is the character just inserted, and does not return a value. You can do just about anything here, but keep in mind that this macro is executed for every keystroke typed, so if you try to get too fancy, you may degrade performance.

NEdit Command Line

```
nedit [-read] [-create] [-line n | +n] [-server]
      [-do command] [-tags file] [-tabs n] [-wrap]
      [-nowrap] [-autowrap] [-autoindent] [-noautoindent]
      [-autosave] [-noautosave] [-rows n] [-columns n]
      [-font font] [-lm languagemode] [-geometry geometry]
      [-iconic] [-noiconic] [-display [host]:server[.screen]
      [-xrm resourcestring] [-svrname name] [-import file]
```

Nirvana Editor (NEdit) Help Documentation

`[-background color] [-foreground color] [-v|-version]
[--] [file...]`

-read

Open the file Read Only regardless of the actual file protection.

-create

Don't warn about file creation when a file doesn't exist.

-line n (or +n)

Go to line number n

-server

Designate this session as an NEdit server, for processing commands from the nc program. nc can be used to interface NEdit to code development environments, mailers, etc., or just as a quick way to open files from the shell command line without starting a new NEdit session.

-do command

Execute an NEdit macro or action. On each file following the `-do` argument on the command line. `-do` is particularly useful from the nc program, where nc `-do` can remotely execute commands in an NEdit `-server` session.

-tags file

Load a file of directions for finding definitions of program subroutines and data objects. The file must be of the format generated by Exuberant Ctags, or the standard Unix ctags command.

-tabs n

Set tab stops every n characters.

-wrap, -nowrap

Wrap long lines at the right edge of the window rather than continuing them past it. (Continuous Wrap mode)

-autowrap, -noautowrap

Wrap long lines when the cursor reaches the right edge of the window by inserting newlines at word boundaries. (Auto Newline Wrap mode)

-autoindent, -noautoindent

Maintain a running indent.

-autosave, -noautosave

Maintain a backup copy of the file being edited under the name `'~filename'`.

Nirvana Editor (NEdit) Help Documentation

-rows n

Default height in characters for an editing window.

-columns n

Default width in characters for an editing window.

-font font (or -fn font)

Font for text being edited (Font for menus and dialogs can be set with `-xrm "*fontList:font"`).

-lm languagemode

Initial language mode used for editing succeeding files.

-geometry geometry (or -g geometry)

The initial size and/or location of editor windows. The argument geometry has the form:

```
[<width>x<height>][+|-][<xoffset>[+|-]<yoffset>]
```

where `<width>` and `<height>` are the desired width and height of the window, and `<xoffset>` and `<yoffset>` are the distance from the edge of the screen to the window, + for top or left, - for bottom or right. `-geometry` can be specified for individual files on the command line.

-iconic, -noiconic

Initial window state for succeeding files.

-display [host]:server[.screen]

The name of the X server to use. `host` specifies the machine, `server` specifies the display server number, and `screen` specifies the screen number. `host` or `screen` can be omitted and default to the local machine, and `screen` 0.

-background color (or -bg color)

User interface background color. (Background color for text can be set separately with `-xrm "nedit.textBgColor: color"` or using the Preferences → Colors dialog).

-foreground color (or -fg color)

User interface foreground color. (Foreground color for text can be set separately with `-xrm "nedit.textFgColor: color"` or using the Preferences → Colors dialog).

-xrm resourcestring

Set the value of an X resource to override a default value (see "[Customizing NEdit](#)").

-svrname name

Nirvana Editor (NEdit) Help Documentation

When starting NEdit in server mode, name the server, such that it responds to requests only when `nc` is given a corresponding `-svrname` argument. By naming servers, you can run several simultaneously, and direct files and commands specifically to any one.

-import file

Loads an additional preferences file on top of the existing defaults saved in your preferences file. To incorporate macros, language modes, and highlight patterns and styles written by other users, run NEdit with `-import <file>`, then re-save your preference file with Preferences → Save Defaults.

-version

Prints out the NEdit version information. The `-V` option is synonymous.

--

Treats all subsequent arguments as file names, even if they start with a dash. This is so NEdit can access files that begin with the dash character.

Client/Server Mode

NEdit can be operated on its own, or as a two-part client/server application. Client/server mode is useful for integrating NEdit with software development environments, mailers, and other programs; or just as a quick way to open files from the shell command line without starting a new NEdit session.

To run NEdit in server mode, type:

```
ncedit -server
```

NEdit can also be started in server mode via the NEdit Client program (**nc**) when no servers are available.

The `nc` program, which is distributed along with NEdit, sends commands to an NEdit server to open files or execute editor actions. It can also be used on files that are already opened.

Listing a file on the `nc` command line means: Open it if it is not already open and bring the window to the front.

`nc` supports the following command line options:

```
nc [-read] [-create]
  [-line n | +n] [-do command] [-lm languagemode]
  [-svrname name] [-svrcmd command]
  [-ask] [-noask] [-timeout seconds]
  [-geometry geometry | -g geometry] [-icon | -iconic]
  [-wait]
  [-V | -version]
  [-xrm resourcestring] [-display [host]:server[.screen]]
  [--] [file...]
```

-read

Open the file read-only regardless of its actual permissions. There is no effect if the file is already open.

Nirvana Editor (NEdit) Help Documentation

-create

Don't warn about file creation when a file doesn't exist.

-line *n*, **+n**

Go to line number *n*. This will also affect files which are already open.

-do *command*

Execute an NEdit macro or action on the file following the `-do` argument on the command line. Note that other files mentioned in the command line are not affected.

If you use this command without a filename, nc would randomly choose one window to focus and execute the macro in.

-ask, **-noask**

Instructs nc to automatically start a server if one is not available. This overrides the X resource ``nc.autoStart'` (see [X Resources](#)).

-svrname *name*

Explicitly instructs nc which server to connect to, an instance of nedit(1) with a corresponding `-svrname` argument. By naming servers, you can run several simultaneously, and direct files and commands specifically to any one.

-svrcmd *command*

The command which nc uses to start an NEdit server. It is also settable via the X resource ``nc.serverCommand'` (see [X Resources](#)). Defaults to "nedit -server".

-lm *language mode*

Initial language mode used.

-geometry *geometry*, **-g** *geometry*

The initial size and/or location of editor windows. See [NEdit Command Line](#) for details.

-icon, **-iconic**

Initial window state.

-display [*<host>*]:*<server>*[.*<screen>*]

The name of the X server to use. See [NEdit Command Line](#) for details.

-timeout *seconds*

Basic time-out period used in communication with an NEdit server. The default is 10 seconds. Also settable via the X resource ``nc.timeOut'`.

Nirvana Editor (NEdit) Help Documentation

Under rare conditions (such as a slow connection), it may be necessary to increase the time-out period. In most cases, the default is fine.

-wait

Instructs nc not to return to the shell until all files given are closed.

Normally, nc returns once the files given in its command line are opened by the server. When this option is given, nc returns only after the last file given in this call is closed.

Note that this option affects all files in the command line, not only the ones following this option.

Note that nc will wait for all files given in the command line, even if the files were already opened.

Command Line Arguments

In typical Unix style, arguments affect the files which follow them on the command line, for example:

```
incorrect:  nc file.c -line 25
correct:   nc -line 25 file.c
```

`-read`, `-create`, and `-line` affect all of the files which follow them on the command line.

The `-do` macro is executed only once, on the next file on the line. `-do` without a file following it on the command line, executes the macro on the first available window (presumably when you give a `-do` command without a corresponding file or window, you intend it to do something independent of the window in which it happens to execute).

The `-wait` option affects all files named in the command line.

Multiple Servers

Sometimes it is useful to have more than one NEdit server running, for example to keep mail and programming work separate. The option, `-svrname`, to both `nedit` and `nc`, allows you to start, and communicate with, separate named servers. A named server responds only to requests with the corresponding `-svrname` argument. If you use ClearCase and are within a ClearCase view, the server name will default to the name of the view (based on the value of the `CLEARCASE_ROOT` environment variable).

Communication

Communication between `nc` and `nedit` is done through the X display. So as long as the X Window System is set up and working properly, `nc` will work properly as well. `nc` uses the `DISPLAY` environment variable, the machine name and your user name to find the appropriate server, meaning, if you have several machines sharing a common file system, `nc` will not be able to find a server that is running on a machine with a different host name, even though it may be perfectly appropriate for editing a given file.

The command which `nc` uses to start an `nedit` server is settable via the X resource `nc.serverCommand`, by default, "`nedit -server`".

Crash Recovery

If a system crash, network failure, X server crash, or program error should happen while you are editing a file, you can still recover most of your work. NEdit maintains a backup file which it updates periodically (every 8 editing operations or 80 characters typed). This file has the same name as the file that you are editing, but with the character `~` (tilde) on Unix or `_` (underscore) on VMS prefixed to the name. To recover a file after a crash, simply rename the file to remove the tilde or underscore character, replacing the older version of the file. (Because several of the Unix shells consider the tilde to be a special character, you may have to prefix the character with a `\<` (backslash) when you move or delete an NEdit backup file.)

Example, to recover the file called "help.c" on Unix type the command:

```
mv \~help.c help.c
```

A minor caveat, is that if the file you were editing was in MS DOS format, the backup file will be in Unix format, and you will need to open the backup file in NEdit and change the file format back to MS DOS via the Save As... dialog (or use the Unix `unix2dos` command outside of NEdit).

Version

NEdit 5.4
Nov 20, 2003

NEdit was written by Mark Edel, Joy Kyriakopoulos, Christopher Conrad, Jim Clark, Arnulfo Zepeda-Navratil, Suresh Ravoora, Tony Balinski, Max Vohlken, Yunliang Yu, Donna Reid, Arne Førlie, Eddy De Greef, Steve LoBasso, Alexander Mai, Scott Tringali, Thorsten Haude, Steve Haehn, Andrew Hood, and Nathaniel Gray.

The regular expression matching routines used in NEdit are adapted (with permission) from original code written by Henry Spencer at the University of Toronto.

Syntax highlighting patterns and smart indent macros were contributed by: Simon T. MacDonald, Maurice Leysens, Matt Majka, Alfred Smeenk, Alain Fargues, Christopher Conrad, Scott Markinson, Konrad Bernloehr, Ivan Herman, Patrice Venant, Christian Denat, Philippe Couton, Max Vohlken, Markus Schwarzenberg, Himanshu Gohel, Steven C. Kapp, Michael Turomsha, John Fieber, Chris Ross, Nathaniel Gray, Joachim Lous, Mike Duigou, Seak Teng-Fong, Joor Loohuis, Mark Jones, and Niek van den Berg.

NEdit sources, executables, additional documentation, and contributed software are available from the NEdit web site at <http://www.nedit.org>.

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

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License in the Help section "[Distribution Policy](#)" for more details.

Distribution Policy

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 675 Mass Ave, Cambridge, MA 02139, USA. Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

Nirvana Editor (NEdit) Help Documentation

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under

the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license

would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Mailing Lists

There are two separate mailing lists for nedit users, and one for developers. Users may post to the developer mailing list to report defects and communicate with the nedit developers. Remember that nedit is entirely a volunteer effort, so please ask questions first to the discussion list, and do your share to answer other users questions as well.

discuss@nedit.org

General discussion, questions and answers among NEdit users and developers.

announce@nedit.org

A low-volume mailing list for announcing new versions.

develop@nedit.org

Communication among and with NEdit developers. Developers should also subscribe to the discuss list.

To subscribe, send mail to <majordomo@nedit.org> with one or more of the following in the body of the message:

```
subscribe announce
subscribe discuss
subscribe develop
```

Problems/Defects

Solutions to Common Problems

For a much more comprehensive list of common problems and solutions, see the NEdit FAQ. The latest version of the FAQ can always be found on the NEdit web site at:

<http://www.nedit.org>.

P: No files are shown in the "Files" list in the Open... dialog.

S: When you use the "Filter" field, include the file specification or a complete directory specification, including the trailing "/" on Unix. (See Help in the Open... dialog).

P: Find Again and Replace Again don't continue in the same direction as the original Find or Replace.

S: Find Again and Replace Again don't use the direction of the original search. The Shift key controls the direction: Ctrl+G means forward, Shift+Ctrl+G means backward.

P: Preferences specified in the Preferences menu don't seem to get saved when I select Save Defaults.

S: NEdit has two kinds of preferences: 1) per-window preferences, in the Preferences menu, and 2) default settings for preferences in newly created windows, in the Default Settings sub-menu of the Preferences menu. Per-window preferences are not saved by Save Defaults, only Default Settings.

P: Columns and indentation don't line up.

S: NEdit is using a proportional width font. Set the font to a fixed style (see Preferences menu).

P: NEdit performs poorly on very large files.

S: Turn off Incremental Backup. With Incremental Backup on, NEdit periodically writes a full copy of the file to disk.

P: Commands added to the Shell Commands menu (Unix only) don't output anything until they are finished executing.

S: If the command output is directed to a dialog, or the input is from a selection, output is collected together and held until the command completes. De-select both of the options and the output will be shown incrementally as the command executes.

P: Dialogs don't automatically get keyboard focus when they pop up.

S: Most X Window managers allow you to choose between two categories of keyboard focus models: pointer focus, and explicit focus. Pointer focus means that as you move the mouse around the screen, the window under the mouse automatically gets the keyboard focus. NEdit users who use this focus model should set "Popups Under Pointer" in the Default Settings sub menu of the preferences menu in NEdit. Users with the explicit focus model, in some cases, may have problems with certain dialogs, such as Find and Replace. In MWM this is caused by the mwm resource startupKeyFocus being set to False (generally a bad choice for explicit focus users). NCDwm users should use the focus model "click" instead of "explicit", again, unless you have set it that way to correct specific problems, this is the appropriate setting for most explicit focus users.

P: The Backspace key doesn't work, or deletes forward rather than backward.

S: While this is an X/Motif binding problem, and should be solved outside of NEdit in the Motif virtual binding layer (or possibly xmodmap or translations), NEdit provides an out. If you set the resource: nedit.remapDeleteKey to True, NEdit will forcibly map the delete key to backspace. The default setting of this resource recently changed, so users who have been depending on this remapping will now have to set it explicitly (or fix their bindings).

P: NEdit crashes when I try to paste text in to a text field in a dialog (like Find or Replace) on my SunOS system.

S: On many SunOS systems, you have to set up an nls directory before various inter-client communication features of Motif will function properly. There are instructions in README.sun in /pub/v5_0_2/individual/README.sun on ftp.nedit.org, as well as a tar file containing a complete nls directory: ftp://ftp.nedit.org/pub/v5_0_2/nls.tar. README.sun contains directions for setting up an nls directory, which is required by Motif for handling copy and paste to Motif text fields.

Known Defects

Below is the list of known defects which affect NEdit. The defects your copy of NEdit will exhibit depend on which system you are running and with which Motif libraries it was built. Note that there are now Motif 1.2 and/or 2.0 libraries available on ALL supported platforms, and as you can see below there are far fewer defects in Motif 1.2, so it is in your best interest to upgrade your system.

All Versions

DEFECT

Operations between rectangular selections on overlapping lines do nothing.

Work Around

None. These operations are very complicated and rarely used.

DEFECT

Cut and Paste menu items fail, or possibly crash, for very large (multi-megabyte) selections.

Work Around

Use selection copy (middle mouse button click) for transferring larger quantities of data. Cut and Paste save the copied text in server memory, which is usually limited.

Reporting Defects

Submit bugs through the web at:

http://sf.net/tracker/?func=add&group_id=11005&atid=111005

Please include the first few lines from Help > Version, which identifies NEdit's version and other system attributes important for diagnosing your problem.

The NEdit developers subscribe to both discuss@nedit.org and develop@nedit.org, either of which may be used for reporting defects. If you're not sure, or you think the report might be of interest to the general NEdit user community, send the report to discuss@nedit.org. If it's something obvious and boring, like we misspelled "anemometer" in the on-line help, send it to develop@nedit.org. If you don't want to subscribe to the [Mailing Lists](#), please add a note to your mail about cc'ing you on responses.