



WinEdit Help Index

The Index lists Help topics available for WinEdit. Use the scroll bar to see entries not currently visible in the Help window.

[OverView](#) [Keyboard and Mouse](#)

Menu Commands

[File Menu](#) [Macro Menu](#)
[Edit Menu](#) [Window Menu](#)
[Search Menu](#) [Utility Menu](#)
[Project Menu](#)

Procedures

[Changing Printers and Printer Options](#)
[Compiling \(Running other programs\)](#)
[Control Bar](#)
[Editing Text](#)
[Working With Multiple Documents](#)
[Extended Help](#)
[Finding Text](#)
[Printing Documents](#)
[Saving Documents](#)
[Setting Preferences](#)
[Setting Margins, Headers and Footers](#)
[Undo and Redo](#)
[Using Regular Expressions](#)
[Windows 3.1 Support](#)

Customizing WinEdit

[Extension DLL](#) [Utility Menu and WIL Language](#)
[WinEdit Extension API's](#) [WIL Language Commands](#)

WinEdit © 1991, 92 Steve Schauer. All rights reserved

WinEdit Overview

WinEdit is an ASCII text editor capable of editing numerous ASCII text files of an almost unlimited size (limited only by available Windows memory). WinEdit is first and foremost a programmer's editor, with many features designed for creating and maintaining program source code. Build, debug and run your programs directly from WinEdit with the ability to view any compiler errors or warnings and the corresponding source code.

As an ASCII text editor, WinEdit allows you to open numerous text files at once, print half sized "two-up" pages side by side in landscape orientation, print headers and footer text (document name, date and time, page number), merge files together, and word wrap your text to the size of the window (word wrap).

Wilson WindowWare, Inc.
2701 California Ave SW #212
Seattle, WA 98116 USA

Orders: (800) 762-8383
Support: (206) 937-9335
Fax: (206) 935-7129

Microsoft® Windows is a Trademark of Microsoft Corporation.

WinEdit Keyboard and Mouse Commands

Moving the Insertion Point

<u>Key(s)</u>	<u>Function</u>
Up Arrow	Moves up one line.
Down Arrow	Moves down one line.
Right Arrow	Moves right one character.
Left Arrow	Moves left one character.
CTRL+Right Arrow	Moves right one word (insertion point is positioned at the beginning of the next word).
CTRL+Left Arrow	Moves left one word (insertion point is positioned at the beginning of the previous word).
Home	Moves to the beginning of the line.
End	Moves to the end of the line.
PgUp	Moves the view up one screenful.
PgDn	Moves the view down one screenful.
CTRL+Home	Moves to the beginning of the document.
CTRL+End	Moves to the end of the document.

Selecting Text

<u>Key(s)</u>	<u>Function</u>
SHIFT+Left or Right Arrow	Extends the selection of text one character at a time.
SHIFT+Down or Up	Selects one line of text up or down from the current selection.
SHIFT+Home	Selects text from the insertion point to the beginning of the line.
SHIFT+End	Selects text from the insertion point to the end of the line.
CTRL+SHIFT+Left Arrow	Selects the previous word.
CTRL+SHIFT+Right Arrow	Selects the next word.
CTRL+SHIFT+PgUp	Selects the previous screen of text.
CTRL+SHIFT+PgDn	Selects the next screen of text.
CTRL+SHIFT+Home	Selects text from the insertion point to the beginning of the document.
CTRL+SHIFT+End	Selects text from the insertion point to the end of the document.

Help Keys

<u>Key(s)</u>	<u>Function</u>
F1	WinEdit Help Index
Shift+F1*	<u>Extended Help</u> (Keyword Help)

*The cursor needs to be positioned on the WINS SDK function, message or data structure name when pressing SHIFT+F1.

Other Keys

Press the **Tab** key to insert a number of spaces and bring the insertion point to the next tab stop. The number of spaces inserted when the tab key is pressed is configurable in File Preferences (choose any value from 1 to 12). For example if the "Tab Size" is set to 3 in File Preferences, pressing the Tab key will advance the cursor three spaces to the right.

The **SHIFT+Tab** key combination moves the current position back to the previous tab stop (to the left). For example if the "Tab Size" is set to 3 in File Preferences, pressing the SHIFT+Tab key combination will move the cursor three spaces/positions to the left.

If more than one line is selected, the Tab and SHIFT+Tab keys will shift every line in the selection forwards (Tab) or backwards (SHIFT+Tab) by one tab stop.

Mouse Operations

Click the Left mouse button on Control Bar along the top of the WinEdit screen and drag your mouse to "tear" the Control Bar from the top of the window. The Control Bar can be resized or moved anywhere on the screen. See [Control Bar](#) for more information.

Click the Right mouse button anywhere on the document window and a [Popup menu](#) will appear with a number of commands (such as Open, Find, Save and Next Error).

Hold the SHIFT key and click the Right mouse button on any Windows SDK function, message, or data structure name and WinEdit will access the SDKWIN.HLP topic for that item.

Double click the Left mouse button over a word to select the word.

Double click the Left mouse on message area of the status bar (the area to the right of the INS/OVR indicator) to jump to the Next Error message.

Double click the Left mouse button on word "Line" in the status bar to bring up the Goto to Line box.

Double click the Left mouse button on "INS" or "OVR" in the status bar to toggle between insert and overwrite mode.

Running Multiple Instances of WinEdit

When starting a new copy of WinEdit, the active copy will be reactivated. If an associated file is double clicked, the current copy will load that file. WinEdit will only allow a single instance, unless the following entry is added to the WINEDIT.INI file:

```
MULTIPLEINST=1
```

The WINEDIT.INI file is a text file located in your Windows directory.

WinEdit Menus

To get help on a particular menu, choose the appropriate top level menu title:

File Menu

Edit Menu

Search Menu

Project Menu

Macro Menu

Utility Menu

Window Menu

WinEdit Procedures

[Changing Printers and Printer Options](#)

[Compiling \(Running other programs\)](#)

[Control Bar](#)

[Editing Text](#)

[Working With Multiple Documents](#)

[Extended Help](#)

[Finding Text](#)

[Printing Documents](#)

[Saving Documents](#)

[Setting Preferences](#)

[Setting Margins, Headers and Footers](#)

[Undo and Redo](#)

[Using Regular Expressions](#)

[Windows 3.1 Support](#)

File Menu Commands

New

Opens a new untitled document window. Existing documents will not be closed when opening a new document.

Open...

Opens a new window with the contents of an existing document/file. List the files of a certain type by selecting one of the following from the lower list box of file types:

- Source Files (*.c;*.h;*.rc;*.def)

- Text Files (*.txt)

- All Files (*.*)

WinEdit can open an ASCII text file as large as available Windows memory. Select the appropriate drive and directory, select a file to open and choose the OK button. The default directory for the File Open command is set in the Project Management dialog (choose Configure from the Project menu). Once a file is opened, the full path and filename is displayed in the caption bar of the window.

Merge...

To merge in the contents of another file into the active windows/document, position your cursor at the location where you would like the text from another file to appear. Choose Merge from the File menu, select a filename and choose the OK button to merge in the text. WinEdit will merge the contents of the file you selected starting on the line just below your insertion point.

Previous Files...

Choose Previous Files to quickly open a file that you previously edited. WinEdit remembers the last 20 files that you have had open and lists these files in the Reopen File dialog. Double click on a filename in the list box or select the file and choose the OK button to open the file. If the you wish to open is not listed, choose the Open Now button to access the standard File Open dialog.

Close

To close the document window before you, select Close from the File menu or double click on the close bar for the active window. If the document has changed since you last saved the file, WinEdit will prompt you to save changes to the file.

Save

Saves the contents of the current window to disk. If the document is UNTITLED, WinEdit prompts you for a document name. In the Save File As dialog, provide a File Name with an extension (WinEdit does not automatically add a file extension to your filename). The file remains open after it is saved to disk.

Save As...

Choose the Save As option to save the contents of the active window to a new or different filename. With the Save As command you can save FILE1.TXT as FILE2.TXT, leaving FILE1.TXT on disk as previously saved with the latest changes written to FILE2.TXT.

Print

Prints the current document using the print settings entered in the Page Setup dialog box.

Page Setup...

Allows you to set the margins, header and/or footer text, select a printer font, and choose a page layout (one portrait page of text per page or two pages in a landscape orientation).

Printer Setup...

Before printing from WinEdit, choose Printer Setup to access printer options for the active Windows printer driver.

Preferences...

Allows you to choose the screen font WinEdit uses for displaying text in the document window, the tab size (number of spaces per tab), whether the ControlBar is shown, and startup window size and an optional message box sound.

Exit

Closes all open windows and exits the WinEdit program. If there are any unsaved files, WinEdit prompts you to save each file before exiting. If you intend to close all of the open windows/documents and not exit WinEdit, then choose Close All from the Window menu.

Edit Menu Commands

Undo

Allows you to "undo" previous editing actions. WinEdit can undo the following edits:

- Inserting a character.
- Deleting a character.
- Cutting a selection.
- Pasting a selection.

WinEdit can undo the last 2000 editing actions. Press CTRL+Z to undo the last editing action.

Redo

Allows you to reverse any Undo command. If you undo an editing action by mistake, you can "redo" the edit. Press SHIFT+CTRL+Z to redo the last editing action.

Cut

Removes the current selection (highlighted text) from the document and places it on the Windows clipboard. You can then paste the contents of the clipboard at another position in the document, into a new document, or into another Windows application.

Copy

Places a copy of the current selection (highlighted text) on the Windows clipboard without removing it from the document. You can then paste the contents of the clipboard at another position in the document, into a new document, or into another Windows application.

Paste

Inserts the text from the Windows clipboard into the document at the current insertion point.

Delete

Removes the current selection from the document without changing the contents of the clipboard. If there is no selection, the character to the right of the insertion point is deleted.

Insert Mode

When Insert Mode is selected from the Edit menu, text is inserted at the current insertion position (if there is text to the right of the insertion position the text is pushed to the right as you insert text). When Insert Mode is not selected, WinEdit is in "Overtyping" mode. When in Overtyping mode, text to the right of the insertion position is overwritten with the new text that is typed. The status line at the bottom of the WinEdit screen, indicates the current editing status:

- OVR - Overtyping is active
- INS - Insert mode is active

The Insert or INS key toggles the editing mode from OVR to INS and back.

Word Wrap

When Word Wrap is selected from the Edit menu, any text you enter will automatically wrap to the next line when the end of the line is reached (the end of the line is determined by the width of the document window). Turning Word Wrap "on" does not affect the word wrap of existing text, but rather only new text that is typed into the window.

Column Block (ALT+C)

To select a column of text, turn the Column Break option on (press ALT+C) prior to selecting text. You can then select text using the mouse or by holding down the Shift key and using the arrow keys on the keyboard to extend the selection. Column select allows you to select a portion of a line for multiple lines, i.e. columns 5-10 for lines 100-120. The status bar at the bottom of the WinEdit window displays the column and line number for the cursor position (i.e. Line 6 Col 16).

Select All

Selects all of the text in the document window.

Search Menu Commands

Find...

Searches for text in a document.

You can specify these options:

- * Find Type the text you want to find.
- * Match Upper/Lower Case Select this box to match the upper and lower case exactly.
- * Forward Search forward in the document starting at the insertion point.
- * Backward Search backward in the document starting at the insertion point.

Repeat Last Find

Allows you to repeat the last search using the same options as the previous search, without opening the Find dialog box again. The quick key combination to repeat the last search/find is CTRL+F5.

Change...

Searches for text within a document and replaces the found text with text you specify (essentially a search and replace function).

You can specify the following options:

- * Find Type the text you want to find in the document.
- * Replace with Type the text you want to insert in place of the found text.
- * Match case Select this box to match the upper and lower case exactly.
- * Search backwards Search backward through the document starting at the insertion point.
- * Confirm before changing When the search text is found, you will be asked if you want to change the occurrence with the replacement text (choose Yes, No or Cancel).
- * Change All When this option is selected, WinEdit will start at the current cursor position and search the entire document. If the "Confirm before changing" box is selected, you will be asked if you want to change the occurrence with the replacement text (choose Yes, No or Cancel).

Next Error

Prev Error

If any warning or error messages have been captured from the output of one of the Project Menu items, the Next and Prev Error menu selections allow you to review the error messages and the corresponding source code. Double clicking with the mouse on message area of the status bar (the area to the right of the INS/OVR indicator) is equivalent to choosing Next Error from the Search menu.

View Compiler Output

If any of the options in the Project Management dialog (choose Configure... from the Project menu) were set to "capture output", the View Compiler Output command will open a new document window with the contents of the captured output file.

Go to line...

Choose this menu item to jump to a particular line number in your document. After choosing "Go to line", type the appropriate line number in the "Go to line" box on the status bar. Press the Enter key and WinEdit will accept the number and move the cursor to the beginning of the indicated line. By default the current line number is displayed in the "Go to line" box on the status bar. There are three ways to access the "Goto box" on the status bar:

Press ALT+G on the keyboard.
Double click on the line and column text on the status bar.
Choose "Go to line..." from the Search menu.

Related Topic:

[Using Regular Expressions](#)

Project Menu Commands

The commands on this menu allow you to run other programs from within WinEdit. Before compiling your program using the choose the Configure menu command to enter the necessary commands to run the program/compiler. Select the Capture Output box and WinEdit will run the program you configure and save its output (choose View Compiler Output from the Search menu to view compiler errors). When the compilation (Build or Rebuild) finishes, WinEdit will ask if you wish to review any warning or error messages, along with the corresponding source code.

%f = file name

%n = base name with no extension

%e = file extension only

Compile command: `tee.com cl -c -AM -W4 -Zps -Od -DNOCOMM %f`

Make command: `tee.com nmk.com %n`

Rebuild command: `tee.com nmk.com /a %n`

Debug command: `cvw %n`

Execute command: `%n`

TEE.COM is included with WinEdit and is the DOS equivalent of the UNIX TEE. WinEdit uses TEE.COM to redirect the stdout and stderr to a file while also echoing the information to the screen.

Note: WinEdit constructs a batch file to execute from DOS when you choose to capture output. For this reason, when running a Windows application from the Run menu, do not choose to capture the output.

Related Topics:

[WinEdit Project Files](#)

[Compiling \(Running other programs\)](#)

Macro Menu Commands

The macro menu includes a macro recorder (Record on) and a listing of your recorded macros. To turn on the macro recorder press ALT+R from the keyboard or choose Record on from the Macro menu. The command name for the Macro recorder changes to Record off when the macro recorder is on (also note that the text "Recording macro" text appears in the lower right corner of the status bar). While the recorder is "on", WinEdit will record your keystrokes so that they can later be assigned to a key for quick playback. To turn off the recorder once your macro is complete, choose Record off from the Macro menu or press ALT+R.

Once the recorder is turned off a dialog will appear prompting you for information about the recorded macro. WinEdit prompts you for a name for the macro. The name you enter will be listed under the macro menu along with its corresponding ALT key combination. The key combinations that can be assigned are ALT+1 - ALT+10.

Example of a recorded macro to delete the current line:

1. Turn on the macro recorder by pressing ALT+R or choose "Record on" from the Macro menu.
2. Press the Home key on the keyboard to move to the beginning of the line.
3. Press SHIFT+END to highlight from the beginning of the line to the end.
4. Press the Del or Delete key.
5. Choose "Record off" from the Macro menu or press ALT+R.
6. Choose a number from 1 to 10 to indicate the ALT key combination for the macro (i.e. ALT+1).
7. Type a name for the macro such as "Delete Line" and choose the OK button.
8. Drop down the macro menu (ALT+M) to see the newly added recorded macro.

Note: The macro recorder can only record keystrokes and not mouse movements.

Window Menu Commands

Tile

Choose Tile from the Windows menu to arrange all of the open windows on the screen so that a portion of each windows can be seen.

Cascade

Choose Cascade to arrange all of the open windows in a stack. When this is done the title bar for each window is visible so that the window can be made active by clicking on the title bar.

Next

Choose Next to change the active window to the next open document window.

Arrange Icons

WinEdit windows that have been minimized appear at the bottom of the screen as an icon. Arrange Icons will place the document window icons along the bottom of the window in rows left to right.

Close All

Closes all open document windows. If changes have been made to a document since it was last saved, you will be prompted to save changes before WinEdit closes the file.

Document Names

Each open window is listed by name at the bottom of the Window menu. Choose a window name and the active window will change so that the selected window will become the active document window.

Utility Menu Commands

The Utility Menu is a custom menu created with WIL commands. For complete help on using the WIL language, consult the WIL.HLP help file.

The menu items on the Utility Menu and the functions behind them are totally configurable. The WINEDIT.MNU file contains the menu item text and the WIL language functions behind them. To edit this file and thus edit the Utility Menu, choose "Edit Utility Menu" from the Utility Menu.

Related Topic:

[WIL Commands](#)

Changing Printers and Printer Options

Select **Printer Setup** from the File menu to change settings in the printer setup dialog for your installed Windows printer drivers. Select a printer driver and choose the Setup button to access the printer driver options (the resulting dialog).

Select **Page Setup** from the File menu to change the following WinEdit page settings:

- Margins
- Header and/or footer text
- The printer font
- Page layout (one or two pages up)

WinEdit will remember your page settings from session to session.

Related Topics:

[Printing Documents](#)

[Setting Margins, Headers and Footers](#)

Compiling (Running other programs)

The first five commands on the Project menu are user-configurable commands to execute another program. You may configure these commands to execute any **.EXE** or **.BAT** program by typing the command text in the appropriate **Configure...** edit box..

For example a sample compile line may read:

```
tee.com cl -c -AM -W4 -Zps -Od -DNOCOMM %f
```

If the program supports DOS redirection (as most compilers and linkers do) you can select the **Capture Output** box to have WinEdit capture the program's output in a file. When the program has executed, WinEdit will allow you to review any messages generated, along with the corresponding source code.

The following wildcards are provided so that filenames in the Project Management dialog do not need to be changed when compiling different files:

%f = file name

%n = base name, no extension

%e = file extension only

If you select the Capture Output box, WinEdit will run the program you configure and save its output. When the program finishes, WinEdit will ask if you wish to review any warning or error messages, along with the corresponding source code.

WinEdit constructs a batch file to execute from DOS when you choose to capture output. For this reason, when running a Windows application from the Run menu, do not choose to capture the output. In this case, just indicate the EXE to run in the Execute command edit box (such as Cardfile). To run the exe listed in the Execute command edit box, do one of the following:

1. Choose the traffic light button on the Control Bar.
2. Press ALT+F7 from the keyboard.
3. Choose Execute from the Project menu.

Related Topics:

[WinEdit Project Files](#)

[Project Menu Commands](#)

Control Bar

The Control Bar allows you to access some frequently used commands by clicking on an icon button along the top of the window. The function of the buttons left to right is as follows:

Icon Buttons



Access the File Open dialog.



Saves the contents of the document window to disk.



Reopen a file from a listing of the last open files.



Prints the contents of the document window.



Restores the previous editing action.



Restores the previous undo action.



Cuts your selected text to the clipboard.



Copy your selected text to the clipboard.



Pastes the contents of the clipboard at the cursor location.



Finds specified text in the document.



Finds text in the document using the previously entered search string.



Allows you to change text (search and replace).



Compile.



Make.



Rebuild. (Visible only on SuperVGA or above resolution displays)



Debug.



Executes your program from WinEdit.



Previous Error.



Next Error.



Keyword Help (Extended Help).

The Control Bar along the top of the WinEdit screen can be resized and repositioned anywhere on the screen. Click the left mouse button on the Control Bar and pull downward to "tear" the Control Bar from the top of the window (the mouse cursor will change to a box with the text TEAR). Let go of the mouse button and the Control Bar is now floating on top of the WinEdit screen. Position your mouse over the edge of the bar and you can resize the bar just as you can document windows. The Control Bar can be docked along any window edge. So if you'd like you can "dock" the control bar on the left of the WinEdit window. To dock the bar, click in the middle of the Control Bar and move your cursor to the middle of the top, left, right or bottom window edge. Move your mouse cursor until the cursor changes from "TEAR" to "DOCK". Once your cursor reads "DOCK" release the left mouse button to dock the Control Bar along the window edge.

You can turn the Control Bar on or off at any time from the File Preferences menu (mark or unmark the Show Control Bar check box).

Editing Text

To "copy and paste", or "cut and paste" the selected text, do the following:

1. Select the text to copy or cut.
2. Choose **Copy** from the Edit menu to copy the selected text to the clipboard. Or choose **Cut** from the Edit menu to cut the text to the clipboard.
3. Move the insertion point where you want the text to appear. Or if you want to replace a section of text with the contents of the clipboard, select the text in the document that you want replaced.
4. Choose **Paste** from the Edit menu.

Editing Shortcuts:

Keypad + (Plus)

Copies the current line to the clipboard if nothing is selected or if there is a selection, the + key functions the same as Edit Copy (CTRL+INS).

Keypad - (Minus)

Cuts the current line to the clipboard if nothing is selected or if there is a selection, the - key is the same as Edit Cut (SHIFT+DEL).

Deleting Text

To delete text without sending it to the Windows clipboard, do the following:

1. Select the text to delete.
2. Choose **Delete** from the Edit menu or press the **Del** key from the keyboard.

If no text is selected, Edit Delete will delete the character to the right of the cursor. To delete characters to the left of the cursor use the Backspace key.

Tab and SHIFT Tab

Press the Tab key to insert a number of spaces and bring the cursor to the next tab stop. The number of spaces inserted when the tab key is pressed is configurable in File Preferences (choose any value from 1 to 12). For example if the "Tab Size" is set to 3 in File Preferences, pressing the Tab key will advance the cursor three spaces to the right.

The SHIFT+Tab key combination moves the current position back to the previous tab stop (to the left). For example if the "Tab Size" is set to 3 in File Preferences, pressing the SHIFT+Tab key combination will move the cursor three spaces/positions to the left.

If more than one line is selected, the Tab and SHIFT+Tab keys will shift every line in the selection forwards (Tab) or backwards (SHIFT+Tab) by one tab stop.

Working With Multiple Documents

WinEdit allows you to open multiple documents and switch back and forth from document to document. To open a new document, choose **New** from the File menu and a new, untitled document window will appear. To load an existing document/file, choose **Open** from the File menu. Choose the file type you would like to open from the list box labeled "List Files of Type:".

There are selections for source files (*.c;*.h;*.rc;*.def), text files (*.txt) and all files regardless of file extensions (*.*). Change to the appropriate drive and directory and double click on the file to open or select the file name and choose the OK button.

Once you have several windows/files open, you can view a list of the open windows by pressing ALT+W. The open files (drives and subdirectory paths included) are listed at the bottom of the Window menu. The active window is indicated by the check mark before the file name. Choose any of the windows listed to change the active window.

Additionally, **Tile** and **Cascade** are available from the Windows menu to change the arrangement of the windows. Choose Cascade to arrange all of the open windows in a stack. When this is done the title bar for each window is visible so that the window can be made active by clicking on the title bar. Choose Tile from the Windows menu to arrange all of the open windows on the screen so that a portion of each windows can be seen.

All of the document windows can also be sized and minimized. To size the document window, move the mouse over a window edge so that the mouse cursor changes to a double sided arrow. Now, click the drag the mouse to change the size of the window. To minimize a document window, click on the down arrow in the upper right corner of the document window. The windows will appear at the bottom of the WinEdit screen as a icon (appears as a white piece of paper). To restore the document window back to its original size, double click on the icon.

Extended Help

Press SHIFT+F1 or hold the SHIFT key and click the *Right* mouse button on any Windows SDK function, message, or data structure name and WinEdit will access the SDKWIN.HLP topic for that item.

For SDK Help, the WinHelp file SDKWIN.HLP must be in either the current directory, the Windows directory, or a directory listed in your DOS PATH statement.

If the selected word is not a valid Windows SDK topic, WinEdit will pass the word to QH.EXE (Microsoft's QuickHelp program, supplied with most Microsoft language products). This will allow you online access to any language or library keyword covered in the QuickHelp database.

For QuickHelp, QH.EXE must be in either the current directory, the Windows directory, or a directory listed in your PATH statement. In addition, you must have an environment variable "HELPPFILES=" defined which tells QH.EXE where to look to find the appropriate QuickHelp database files.

Finding Text

Choose **Find** from the Search menu to search for text within the active document.

You can specify the following options:

- * Find Type the text you want to find.
- * Match Upper/Lower Case Select this box to match the upper and lower case exactly.
- * Forward Search forward in the document starting at the insertion point.
- * Backward Search backward in the document starting at the insertion point.

Choose [Repeat Last Find](#) (or press CTRL+F5) to repeat the last search using the same options as the previous search, without opening the Find dialog box again.

Choose [Change](#) from the Search menu to search for text in a document and replace the found text with text you specify.

You can specify the following options:

- * Find Type the text you want to find in the document.
- * Replace with Type the text you want to insert in place of the found text.
- * Match case Select this box to match the upper and lower case exactly.
- * Search backwards Search backward through the document starting at the insertion point.
- * Confirm before changing When the search text is found, you will be asked if you want to change the occurrence with the replacement text (choose Yes, No or Cancel).
- * Change All When this option is selected, WinEdit will start at the current cursor position and search the entire document. If the "Confirm before changing" box is selected, you will be asked if you want to change the occurrence with the replacement text (choose Yes, No or Cancel).

Related Topic:

[Using Regular Expressions](#)

Using Regular Expressions

A regular expression is a search or replace string that uses special characters to match text patterns. WinEdit supports UNIX style regular expressions.

When WinEdit conducts a search using regular expressions, it must check character by character in your text. For this reason, searches using regular expressions are slower than regular searches.

The following table describes the regular expression characters recognized by WinEdit.

Expression	Description
\	<i>Escape.</i> WinEdit will ignore any special meaning of the character that follows the Escape expression. Use the Escape if you need to search for a literal character that matches a regular expression character.
.	<i>Wild Card.</i> Matches any character. For example, the expression 'X.X' will match 'XaX', 'XbX', and 'XcX', but not 'XaaX'.
^	<i>Beginning Of Line.</i> The expression matches only if it occurs at the beginning of a line. For example, '^for' matches the text 'for' only when it occurs at the beginning of a line.
\$	<i>End Of Line.</i> The expression matches only if it occurs at the end of a line. For example, '(void)\$' matches the text '(void)' only when it occurs at the end of a line.
[]	<i>Character Class.</i> The expression matches any character in the class specified within the brackets. Use a dash (-) to specify a range of character values. For example, '[a-zA-Z0-9]' matches any letter or number, and '[xyz]' matches 'x', 'y', or 'z'.
[^]	<i>Inverse Class.</i> The expression matches any character not specified in the class. For example, '[^a-zA-Z]' matches any character that is not a letter.
*	<i>Repeat Operator.</i> Matches zero or more occurrences of the character that precedes the '*'. For example, 'XY*X' matches 'XX', 'XYX', and 'XYYX'.
+	<i>Repeat Operator.</i> Matches one or more occurrences of the character that precedes the '+'. For example, 'XY+X' matches 'XYX' and 'XYYX', but not 'XX'.

Related Topic:

[Finding Text](#)

Windows 3.1 Support

"Drag and Drop"

WinEdit is fully compatible with Windows version 3.1. WinEdit version 2.0 adds support for drag and drop from File Manager. To open files in WinEdit, simply drag and drop one or more files from the Windows File Manager onto WinEdit. Drag and drop from File Manager is functional when WinEdit is either minimized or maximized.

"Sounds"

Whenever a Message Box comes up, WinEdit plays the WAVE file corresponding to the Message Box icon under Multimedia Windows or the Asterisk event under Windows 3.1. Under Windows Win 3.0 the standard beep will play. "Warning Sounds" is an option in File Preferences dialog and can be disabled if you'd like. There is also a WinEdit Startup event that can be configured for any WAVE file when operating under Windows 3.1 or Multimedia Windows (sound card required).

Printing Documents

Choose **Print** from the File menu to send the text of the current document to the active printer. All print options such as the layout (one up portrait printing or two page landscape printing) and printer font, are set in the Page Setup dialog (accessible from the File menu). The default printer selection is made in the Windows Control Panel Printers section.

When choosing Print, the entire contents of the document file is sent to the printer. To print only a selection of text, do the following:

1. Select the text to print.
2. Choose Copy from the Edit menu.
3. Choose New from the File menu.
4. Choose Paste from the Edit menu to paste the contents of the clipboard.
5. Select the printer button on the Control Bar or choose Print from the File menu.

Related Topics:

[Changing Printers and Printer Options](#)
[Setting Margins, Headers and Footers](#)

Saving Documents

To save a document to disk under the current file name (the filename appears in the title bar for the document window), choose **Save** from the File menu. If the file has not been saved before, WinEdit will prompt for a file name. To save a document to a new name and/or location, choose **Save As** from the File menu.

When a file is saved in WinEdit, the previous version of the document is renamed to the with a .BAK file extension. For example, if you make changes to FILENAME.TXT and choose to save the file, the previous version of the file is renamed to FILENAME.BAK while the new changes are saved to FILENAME.TXT. The backup file (FILENAME.BAK) is saved to the same directory as FILENAME.TXT.

Setting Preferences

The WinEdit program preferences are accessible by choosing **Preferences** from the File menu.

To change the screen font used to display document text, choose the button labeled Screen Font. In the resulting dialog choose a font name from the listing of fixed spaced screen fonts on your system. In addition to the typeface, there are four styles to choose from: Regular, Italic, Bold and Bold Italic. Select a style a point size for your on screen text.

The Tab Size list box allows you to set the number of spaces you would like to be inserted into your document, each time you press the tab key. Values range from one space to twelve spaces per tab.

The "Reopen last file at startup" is helpful if you usually start working were you left off in your last WinEdit session. If this option is selected, the last file open in WinEdit will be opened automatically the next time WinEdit is started. If this option is not selected, WinEdit will open with no windows open (use the File New or File Open command to begin working in WinEdit).

The "Zoom window at startup" option determines the size of the document window on startup of WinEdit. If the "Zoom" option is selected, then the WinEdit program will start with the document window zoomed out so that it fills the WinEdit program area. If the "Zoom" option is not selected in File Preferences, then the WinEdit document window will not fill the WinEdit program area.

The Show Control Bar option simply toggles the ControlBar of buttons On (if selected) or Off if not selected in File Preferences.

The "Warning sounds" option plays the WAVE file corresponding to the Message Box icon under Multimedia Windows or Windows 3.1, whenever a Message Box pops up such as search text not found. Under Windows version 3.0, the standard system beep will sound.

Setting Margins, Headers and Footers

Headers and Footers

Choose **Page Setup** from the File menu to configure header and footer text. Type the text you wish to appear at the top and bottom of each page.

You can use the following special characters in headers and footers:

- * %f The document name will appear.
- * %d The date and time of the printout will appear.
- * %p The page number will appear.

The default header text is "%f - %d" or Document Name - Date and Time of the printout.

The default footer text is "Page %p" or Page 1.

Changes made to the header and footer text are remembered for the next session of WinEdit.

Margins

Choose **Page Setup** from the File menu to change the margins used for WinEdit's printouts. You can enter the measurements for top, bottom, left, and right margins. The margin values are either in inches or centimeters, depending upon the Measurement setting in the Windows Control Panel International section/icon.

Related Topics:

[Changing Printers and Printer Options](#)

[Printing Documents](#)

Undo and Redo

Undo

Allows you to "undo" previous editing actions. WinEdit can undo the following edits:

- Inserting a character.

- Deleting a character.

- Cutting a selection.

- Pasting a selection.

WinEdit can undo the last 2000 editing actions. Press ALT+Backspace to undo the last editing action or select Undo from the Edit menu.

Redo

Allows you to reverse any Undo command. If you undo an editing action by mistake, you can "redo" the edit. Press CTRL+Backspace to redo the last editing action or choose Redo from the Edit menu.

WinEdit Project Files

Information entered into the Project Management dialog (choose Configure... from the Project menu) can be saved in a private INI file with a **.WPJ** (WinEdit Project File) extension. Choose the **Save...** pushbutton to save the contents of the dialog box in a .WPJ file. The default save path for the WPJ files is the Windows directory. After creating several project files you can load an existing .WPJ file by clicking on the **Open...** pushbutton.

There five edit boxes for your Compile, Make, Rebuild, Debug and Execute command line information. The following wildcards are provided so that filenames in the Project Management dialog do not need to be changed when compiling different files:

%f = file name

%n = base name, no extension

%e = file extension only

If you select the Capture Output box, WinEdit will run the program you configure and save its output. When the program finishes, WinEdit will ask if you wish to review any warning or error messages, along with the corresponding source code. WinEdit constructs a batch file to execute from DOS when you choose to capture output. For this reason, when running a Windows application from the Run menu, do not choose to capture the output.

The Project Name field can be filled with a brief description of the project. The Working Directory field sets the default open and save directories for your project to the path indicated in this field.

Note: The last project file open in WinEdit, will automatically be loaded the next time you start WinEdit. To change to another project file, choose Configure... from the Project menu and choose the Open... pushbutton.

WinEdit Extensions

A WinEdit Extension DLL is a dynamic-link library (DLL) that contains a pre-defined entry point that processes menu commands and notification messages sent by WinEdit. You can redefine WinEdit's menus and accelerators and write new functions which access WinEdit functions directly.

Creating a WinEdit Extension

A WinEdit Extension DLL must be named WE_EXT.DLL and must include a standard entry point, the WE_ExtensionProc function. It must include the WE_EXT.H header file that defines WinEdit messages and structures. WinEdit communicates with the Extension DLL by sending messages to the DLL's WE_ExtensionProc function.

The WE_ExtensionProc function is defined as follows:

```
WE_ExtensionProc(HWND hWnd,      /* WinEdit's window handle */
                 HANDLE hInst,   /* instance identifier      */
                 UINT wParam,    /* command ID              */
                 LONG lParam)    /* additional information   */
```

The hWnd parameter identifies the main WinEdit window. This window handle is used in most of the extension functions, and should also be used as the parent window for any child windows, dialog boxes, or message boxes created.

The hInst parameter is the HINSTANCE of the Extension DLL. This parameter is used when retrieving resources from the DLL.

The wParam parameter contains the message ID, which may be a command ID from a menu or accelerator, a notification message from WinEdit, or a request for information from WinEdit.

The lParam parameter is used in some messages to pass additional information to the Extension DLL.

Loading the Extension

WinEdit searches the current directory and the path for WE_EXT.DLL, and explicitly loads the Extension DLL if it is found. If the DLL is successfully loaded, WinEdit then sends the following notification and request messages to the Extension DLL:

WEN_LOADMENU

This message is a request for a menu handle to be used as the main WinEdit menu. If the Extension DLL returns a handle to a menu as the return value for this message, WinEdit uses that menu. If the Extension DLL returns 0, the standard WinEdit menu is used.

WEN_LOADSHORTMENU

This message is a request for a menu handle to be used as the "No file" menu. WinEdit displays this menu whenever no MDI child windows are open. If the Extension DLL returns a handle to a menu as the return value for this message, WinEdit uses that menu. If the Extension DLL returns 0, the standard WinEdit menu is used.

WEN_LOADACCELS

This message is a request for a handle to an accelerator table. If the Extension DLL returns a handle to an accelerator table as the return value for this message, WinEdit will use that accelerator table. If the Extension DLL returns 0, the standard WinEdit accelerator table is used.

WEN_GETWINDOWMENU

If the Extension DLL returned a menu handle in the WEN_LOADMENU message,

this message will be sent to obtain the handle to the Window popup menu. WinEdit uses this menu handle to append MDI child window names to.

WEN_GETMACROMENU

If the Extension DLL returned a menu handle in the WEN_LOADMENU message, this message will be sent to obtain the handle to the Macro popup menu. WinEdit uses this menu handle to append macro names to.

WEN_INITMENU

This message is sent before showing any drop down menu items. Respond by setting any check marks, graying any inapplicable items, etc.

WEN_MACROSTARTPLAY

This message is sent when a keyboard macro playback is started.

WEN_MACROSTOPPLAY

This message is sent when a keyboard macro playback is stopped.

WEN_MACROSTARTRECORD

This message is sent when a keyboard macro recording is started. The menu item for toggling starting and stopping recording should be changed to "Stop Recording".

WEN_MACROSTOPRECORD

This message is sent when a keyboard macro recording is stopped. The menu item for toggling starting and stopping recording should be changed to "Record Macro".

WEN_RBUTTONDOWN

This message is sent when the right mouse button is clicked in an MDI child window.

WEN_RBUTTONDOWNNC

This message is sent when the right mouse button is clicked while the control key is down in an MDI child window.

WEN_RBUTTONDOWNNS

This message is sent when the right mouse button is clicked while the shift key is down in an MDI child window.

WEN_RBUTTONDOWNNSC

This message is sent when the right mouse button is clicked while the control key and the shift keys are down in an MDI child window.

WEN_END

This message is sent before the DLL is unloaded. Any cleanup processing should be done, such as releasing allocated memory.

Processing Menu Selections

A WinEdit Extension DLL's menu resource can include two levels of menu identifiers. Identifiers in the range of WE_EXTFIRST through WE_EXTLAST (defined in WE_EXT.H) are sent to the Extension DLL for processing. Other identifiers defined in WE_EXT.H that begin with IDM_ are internal WinEdit commands that WinEdit handles without calling the Extension DLL.

When designing menus or accelerators, use the IDM_ identifiers for predefined functions which WinEdit will handle without further processing by your Extension DLL. Use identifiers in the range of WE_EXTFIRST through WE_EXTLAST for functions you define. When the user selects a menu item or presses an accelerator key which is defined with an identifier in that range, WinEdit will pass the message on to the Extension DLL for processing. The following code example shows a simple example of a user-defined

function being called:

```
#define EXT_EXAMPLE WE_EXTFIRST+1
.
.
.
switch (wParam)
{
    case EXT_EXAMPLE:
        return MyFunction();
        break;
```

Initializing the Extension Menu

Whenever the user selects a menu item, WinEdit sends the WEN_INITMENU message to the Extension DLL. The Extension DLL should respond to this message by adding check marks or disabling or enabling items. If the Extension DLL did not load its own menu, it can ignore this message.

More Information

[WinEdit Extension Example](#)

WinEdit Extension Example

The following example shows a minimal WinEdit Extension DLL's WE_ExtensionProc function. The Extension DLL loads its own menus and accelerator table, and defines one new command.

```
#include <windows.h>
#include "we_ext.h"

#define MACROMENU    4    /* position of macro menu (0 based)    */
#define WINDOWMENU  5    /* position of window menu (0 based)    */
#define EXT_EXAMPLE  WE_EXTFIRST+1

UINT FAR PASCAL WE_ExtensionProc(HWND hWnd,    /* WinEdit's window handle */
                                HANDLE hInst, /* instance identifier     */
                                UINT wParam,  /* command ID              */
                                LONG lParam)  /* additional information   */
{
    switch (wParam)
    {
        case WEN_LOADMENU:

            /* This is the menu WinEdit will display when there
             * is at least one document window open. Return NULL
             * to use the default WinEdit menu.
             */
            return (UINT)LoadMenu(hInst, "MyMenu");
            break;

        case WEN_LOADSHORTMENU:

            /* this is the menu WinEdit will display when there
             * are no document windows open. Return NULL
             * to use the default WinEdit menu.
             */
            return (UINT)LoadMenu(hInst, "MyShortMenu");
            break;

        case WEN_LOADACCELS:

            /* To re-define the WinEdit command keys, load your
             * own accelerator table here. Return NULL to
             * use the default WinEdit accelerators.
             */
            return (UINT)LoadAccelerators (hInst,"MyAccels");
            break;

        case WEN_GETWINDOWMENU:

            /* WinEdit needs the handle of the submenu to
             * append MDI document names to. The hWnd parameter
             * is used to send the handle to the main menu.
             * This message will not be sent if you return
             * NULL to the WEN_LOADMENU message.
             */
            return (UINT)GetSubMenu ((HMENU)hWnd, WINDOWMENU);
            break;

        case WEN_GETMACROMENU:

            /* WinEdit needs the handle of the submenu to
             * append macro names to. The hWnd parameter
```

```

    * is used to send the handle to the main menu.
    * This message will not be sent if you return
    * NULL to the WEN_LOADMENU message.
    */
return (UINT)GetSubMenu ((HMENU)hWnd, MACROMENU);
break;

case WEN_END:

    /* WinEdit is shutting down. Do any clean-up processing
    * here.
    */
return TRUE;
break;

case WEN_INITMENU:

    /* This message is sent before showing any drop down
    * menu items. Respond by setting any checkmarks,
    * graying any inapplicable items, etc.
    */
    {
POINT ptStart,ptEnd;
HMENU hCurrentMenu;
UINT wStatus;

hCurrentMenu = GetMenu(hWnd);

    /* if there is a current selection, enable the cut & copy
    * commands.
    */
wStatus = (UINT)edGetSelectionState(hWnd, &ptStart, &ptEnd);
if (!wStatus)
    wStatus = MF_GRAYED;
else
    wStatus = MF_ENABLED;
EnableMenuItem(hCurrentMenu, IDM_EDITCUT, wStatus);
EnableMenuItem(hCurrentMenu, IDM_EDITCOPY, wStatus);

    /* if there is text on the clipboard, enable the paste
    * command.
    */
if (OpenClipboard(hWnd))
    {
    if (IsClipboardFormatAvailable(CF_TEXT)
        || IsClipboardFormatAvailable(CF_OEMTEXT))
        EnableMenuItem(hCurrentMenu, IDM_EDITPASTE, MF_ENABLED);
    else
        EnableMenuItem(hCurrentMenu, IDM_EDITPASTE, MF_GRAYED);
    CloseClipboard();
    }
else
    EnableMenuItem(hCurrentMenu, IDM_EDITPASTE, MF_GRAYED);

    /* set the Undo, Redo, Insert, and WordWrap menu items */
wStatus = (UINT)edGetUndoState(hWnd);
if (!wStatus)
    wStatus = MF_GRAYED;
else
    wStatus = MF_ENABLED;
EnableMenuItem(hCurrentMenu, IDM_EDITUNDO, wStatus);

wStatus = (UINT)edGetRedoState(hWnd);

```

```

    if (!wStatus)
        wStatus = MF_GRAYED;
    else
        wStatus = MF_ENABLED;
    EnableMenuItem(hCurrentMenu, IDM_EDITREDO, wStatus);

    wStatus = (UINT)edGetWordWrapState(hWnd);
    if (!wStatus)
        wStatus = MF_UNCHECKED;
    else
        wStatus = MF_CHECKED;
    CheckMenuItem (hCurrentMenu, IDM_EDITTOGGLEWRAP, MF_BYCOMMAND|wStatus);

    wStatus = (UINT)edGetInsertState(hWnd);
    if (!wStatus)
        wStatus = MF_UNCHECKED;
    else
        wStatus = MF_CHECKED;
    CheckMenuItem (hCurrentMenu, IDM_EDITTOGGLEINS, MF_BYCOMMAND|wStatus);

    return TRUE; /* we handled it, don't return 0 */
    break;
}

/* You can define your own commands in the range
 * WE_EXTFIRST to WE_EXTLAST that can be attached to
 * menu items or accelerators.
 */
case EXT_EXAMPLE:
    MessageBox(hWnd, "Example command", "WinEdit Extension",
        MB_ICONINFORMATION|MB_OK);
    return TRUE;
    break;

default:

    /* return NULL to all messages not processed. */
    break;

} /* end switch (wParam) */

return NULL;
}

```

More Information

WinEdit Extension API's

WinEdit Extension API's

[edAddButton](#)
[edDeleteButton](#)
[edFileList](#)
[edFileNew](#)
[edFileOpen](#)
[edFileMerge](#)
[edFileSave](#)
[edFileSaveAs](#)
[edFilePrint](#)
[edFilePageSetup](#)
[edFilePrinterSetup](#)
[edFileSetPreferences](#)
[edFileExit](#)
[edEditUndo](#)
[edEditRedo](#)
[edEditCut](#)
[edEditCopy](#)
[edEditPaste](#)
[edEditDelete](#)
[edEditToggleIns](#)
[edEditWrap](#)
[edEditSetColumnBlock](#)
[edEditInsertString](#)
[edEditBackspace](#)
[edEditSelectAll](#)
[edEditCopyLine](#)
[edEditCutLine](#)
[edEditGoToLine](#)
[edEditGoToColumn](#)
[edEditBeginningOfLine](#)
[edEditEndOfLine](#)
[edEditBeginningOfFile](#)
[edEditEndOfFile](#)
[edEditDownLine](#)
[edEditUpLine](#)
[edEditLeft](#)
[edEditRight](#)
[edEditPageUp](#)
[edEditPageDown](#)
[edEditWordLeft](#)
[edEditWordRight](#)
[edEditStartSelection](#)
[edEditEndSelection](#)
[edEditClearSelection](#)
[edEditSetBookmark](#)
[edEditGoToBookmark](#)
[edEditTab](#)
[edEditBackTab](#)
[edEditGetCurrentWord](#)
[edSearchFind](#)
[edSearchRepeat](#)
[edSearchChange](#)
[edSearchNextError](#)

edSearchPrevError
edSearchViewOutput
edRunCompile
edRunMake
edRunRebuild
edRunDebug
edRunExecute
edRunCommand
edRunConfigure
edWindowTile
edWindowCascade
edWindowArranageIcons
edWindowMinimize
edWindowMaximize
edWindowRestore
edWindowClose
edWindowsCloseAll
edHelpIndex
edHelpKeyboard
edHelpCommands
edHelpProcedures
edHelpKeyWord
edHelpHelp
edHelpAbout
edGetModifiedStatus
edGetLineNumber
edGetColumnNumber
edGetSelectionMode
edGetUndoState
edGetRedoState
edGetWordWrapState
edGetInsertState
edGetWindowName
edGetChar

SearchRecord

edAddButton

Syntax

```
int FAR PASCAL edAddButton(HWND hWnd, WORD wIcon, WORD wCommand, WORD wPosition)
```

Parameters

HWND hWnd

Identifies the WinEdit window.

WORD wIcon

Identifies which icon to display.

WORD wCommand

The command ID to be called when this button is pressed. This can be any of the IDM_ values for internal WinEdit commands, or the ID of an Extension DLL function.

WORD wPosition

The 0-based position on the control bar for the button to be added.

Return Value

The result is nonzero if the function was successful. Otherwise it is zero.

Comments

edAddButton adds a button to the control bar. The following WinEdit icon IDs are documented in the WE_EXT.H include file:

```
#define IDLEXICON    6    /* main icon          */
#define IDNOTE      2    /* icon for child windows */
#define IDONEUP     4    /* one-up print icon   */
#define IDTWOUP    5    /* two-up print icon   */

#define OPEN        48   /* file open           */
#define SAVE        49   /* file save           */
#define PRINT       50   /* file print          */
#define FIND        51   /* find                */
#define FINDNEXT    52   /* find next           */
#define CHANGE      53   /* change              */
#define NEXT        54   /* next error          */
#define PREV        55   /* prev error          */
#define COMPILE     56   /* compile             */
#define MAKE        57   /* make                */
#define REBUILD     58   /* rebuild             */
#define DEBUGICON   59   /* debug               */
#define EXECUTE     60   /* execute             */
#define CUT         61   /* cut                 */
#define COPY        62   /* copy                */
#define PASTE       63   /* paste               */
#define HELPKEY     64   /* key word help      */
#define UNDO        65   /* undo                */
#define REDO        66   /* redo                */
#define FILELIST    67   /* filelist            */
```

edDeleteButton

Syntax

int FAR PASCAL edDeleteButton(HWND hWnd, WORD wPosition, WORD wCommand);

Parameters

HWND hWnd

Identifies the WinEdit window.

WORD wPosition

The 0-based position on the control bar for the button to be added.

WORD wCommand

The command ID associated with this button.

Return Value

The result is nonzero if the function was successful. Otherwise it is zero.

Comments

The default WinEdit control bar has the following commands:

Position	Command
-----	-----
0	IDM_FILEOPEN
1	IDM_FILESAVE
2	IDM_FILELIST
3	IDM_FILEPRINT
4	IDM_EDITUNDO
5	IDM_EDITREDO
6	IDM_EDITCUT
7	IDM_EDITCOPY
8	IDM_EDITPASTE
9	IDM_SEARCHFIND
10	IDM_SEARCHNEXT
11	IDM_SEARCHCHANGE
12	IDM_COMPILE
13	IDM_MAKE
14	IDM_REBUILD
15	IDM_DEBUG
16	IDM_EXECUTE
17	IDM_SEARCHPREVEERROR
18	IDM_SEARCHNEXTERROR
19	IDM_HELPKEYWORDS

edFileList

Syntax

int FAR PASCAL edFileList(HWND hWnd);

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the function was successful. Otherwise it is zero.

Comments

edFileList brings up the Reopen File dialog box (same as selecting Previous Files from the File menu), allowing the user to pick a file to open from a list of the last 20 previously opened files.

edFileNew

Syntax

int FAR PASCAL edFileNew(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the new window was successfully opened. Otherwise it is zero.

Comments

edFileNew creates a new MDI child window.

edFileOpen

Syntax

int FAR PASCAL edFileOpen(HWND hWnd, LPSTR lpFileName)

Parameters

HWND hWnd

Identifies the WinEdit window.

LPSTR lpFileName

Name of file to open.

Return Value

The result is nonzero if a new window was created and the file was read. Otherwise it is zero.

Comments

edFileOpen creates a new MDI child window and reads an existing file into the window. To open a file without prompting, pass a valid file name to edFileOpen in the lpFileName parameter. If lpFileName is NULL, the File Open dialog box will be used to obtain a file name from the user.

edFileMerge

Syntax

int FAR PASCAL edFileMerge(HWND hWnd, LPSTR lpFileName)

Parameters

HWND hWnd

Identifies the WinEdit window.

LPSTR lpFileName

Name of file to merge into current window. This must be the name of an existing file.

Return Value

The result is nonzero if the file was read. Otherwise it is zero.

Comments

edFileMerge reads an existing file into the active MDI child window. To merge a file without prompting, pass a valid file name to edFileMerge in the lpFileName parameter. If lpFileName is NULL, the File Merge dialog box will be used to obtain a file name from the user.

edFileSave

Syntax

int FAR PASCAL edFileSave(HWND hWnd)

Parameters

HWND hWnd
Identifies the WinEdit window.

Return Value

The result is nonzero if the file was successfully saved. Otherwise it is zero.

Comments

edFileSave saves the file in the currently active MDI child window without prompting.

See Also

[edGetModifiedStatus](#)
[edFileSaveAs](#)

edFileSaveAs

Syntax

int FAR PASCAL edFileSaveAs(HWND hWnd, LPSTR lpFileName)

Parameters

HWND hWnd

Identifies the WinEdit window.

LPSTR lpFileName

Name of file to save.

Return Value

The result is nonzero if the file was successfully saved. Otherwise the return value is zero.

Comments

edFileSaveAs saves the file in the currently active MDI child window. If lpFileName is not NULL, edFileSaveAs saves the file with that name without prompting. If lpFileName is NULL, the File Save As dialog box will be used to obtain a file name from the user.

See Also

[edGetModifiedStatus](#)

[edFileSave](#)

edFilePrint

Syntax

int FAR PASCAL edFilePrint(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the print job file was successful. Otherwise it is zero.

edFilePageSetup

Syntax

int FAR PASCAL edFilePageSetup(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the function was successful. Otherwise it is zero.

Comments

edFilePageSetup brings up the Page Setup dialog box.

edFilePrinterSetup

Syntax

int FAR PASCAL edFilePrinterSetup(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the function was successful. Otherwise it is zero.

Comments

edFilePrinterSetup brings up a dialog box listing all installed printers. The user can choose a printer from the list, which WinEdit will use for all print jobs. The user can also access the printer's Setup dialog box to change printer settings. These changes, if any, are used for the current editing session only and do not change the system wide printer settings.

edFileSetPreferences

Syntax

int FAR PASCAL edFileSetPreferences(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the function was successful. Otherwise it is zero.

Comments

edFileSetPreferences allows the user to set screen font, tab size, and other configuration options through a dialog box. The results are stored in WINEDIT.INI and used in future editing sessions.

edFileExit

Syntax

int FAR PASCAL edFileExit(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

TRUE if the user did not cancel the exit. FALSE otherwise.

Comments

If there are any unsaved files, the user will be prompted to save before closing. The user can cancel the exit operation at that point. If there are no unsaved files, the exit is unconditional.

See Also

[edGetModifiedStatus](#)

[edFileSave](#)

[edFileSaveAs](#)

edEditUndo

Syntax

int FAR PASCAL edEditUndo(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

TRUE if any operation was undone, FALSE otherwise.

See Also

[edGetUndoState](#)

edEditRedo

Syntax

int FAR PASCAL edEditRedo(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

TRUE if any operation was redone, FALSE otherwise.

See Also

[edGetRedoState](#)

edEditCut

Syntax

int FAR PASCAL edEditCut(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

TRUE if any text was cut to the clipboard, FALSE otherwise.

Comments

edEditCut cuts the current selection to the clipboard.

See Also

[edEditCutLine](#)

[edEditDelete](#)

edEditCopy

Syntax

int FAR PASCAL edEditCopy(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

TRUE if any text was copied to the clipboard, FALSE otherwise.

Comments

edEditCopy copies the current selection to the clipboard.

See Also

[edEditCopyLine](#)

edEditPaste

Syntax

int FAR PASCAL edEditPaste(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

TRUE if any text was pasted from the clipboard, FALSE otherwise.

Comments

edEditPaste pastes text from the clipboard into the active MDI child window.

edEditDelete

Syntax

int FAR PASCAL edEditDelete(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

TRUE if any text was deleted, FALSE otherwise.

Comments

edEditDelete deletes either the current selection or, if there is no selection, the character following the current insertion position. The text is deleted and is not copied to the clipboard.

See Also

[edEditCut](#)

[edEditCutLine](#)

edEditToggleIns

Syntax

int FAR PASCAL edEditToggleIns(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

TRUE if the Insert state was changed, FALSE otherwise.

Comments

edEditToggleIns toggles the insert state between Insert and Overtyping modes.

See Also

[edGetInsertState](#)

edEditWrap

Syntax

int FAR PASCAL edEditWrap(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

TRUE if word wrap state was changed, FALSE otherwise.

Comments

edEditWrap toggles the word wrap state on or off.

See Also

[edGetWordWrapState](#)

edEditSetColumnBlock

Syntax

int FAR PASCAL edEditSetColumnBlock(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

TRUE if column block state was set, FALSE otherwise.

Comments

edEditSetColumnBlock enables column block marking for the next block operation.

WinEdit automatically returns to stream block marking after the next block operation.

edEditInsertString

Syntax

int FAR PASCAL edEditInsertString(HWND hWnd, LPSTR lpString)

Parameters

HWND hWnd

Identifies the WinEdit window

LPSTR lpString

Identifies the text to be inserted.

Return Value

TRUE if any text was inserted.

Comments

edEditInsertString inserts lpString at the current insertion position.

edEditBackspace

Syntax

int FAR PASCAL edEditBackspace(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise it is zero.

Comments

edEditBackSpace deletes the character to the left of the current position.

edEditSelectAll

Syntax

int FAR PASCAL edEditSelectAll(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise it is zero.

Comments

edEditSelectAll selects all the text in the active window. The current position is moved to the end of the file.

edEditCopyLine

Syntax

int FAR PASCAL edEditCopyLine(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise it is zero.

Comments

edEditCopyLine copies the current line to the clipboard if there is no selection. If there is a selection, edEditCopyLine calls edEditCopy and copies the current selection to the clipboard.

See Also

[edEditCopy](#)

edEditCutLine

Syntax

int FAR PASCAL edEditCutLine(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise it is zero.

Comments

edEditCutLine cuts the current line to the clipboard if there is no selection. If there is a selection, edEditCutLine calls edEditCut and cuts the current selection to the clipboard.

See Also

[edEditCut](#)

edEditGoToLine

Syntax

int FAR PASCAL edEditGoToLine(HWND hWnd, int iLineNo)

Parameters

HWND hWnd

Identifies the WinEdit window

int iLineNo

Identifies the line number to go to.

Return Value

The result is nonzero if the current position was changed to iLineNo. Otherwise it is zero.

Comments

edEditGoToLine moves the current position to the line identified by the iLineNo parameter. If iLineNo is greater than the last line in the file, the current position is moved to the last line in the file.

edEditGoToColumn

Syntax

int FAR PASCAL edEditGoToColumn(HWND hWnd, int iColNo)

Parameters

HWND hWnd

Identifies the WinEdit window.

int iColNo

Identifies the column number to go to.

Return Value

The result is nonzero if the current position was changed to iColNo. Otherwise it is zero.

Comments

edEditGoToColumn moves the current position to the column identified by iColNo.

edEditBeginningOfLine

Syntax

int FAR PASCAL edEditBeginningOfLine(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise it is zero.

Comments

edEditBeginningOfLine moves the current position to Column 1.

edEditEndOfLine

Syntax

int FAR PASCAL edEditEndOfLine(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise it is zero.

Comments

edEditEndOfLine moves the current position to the column following the last text character in the current line.

edEditBeginningOfFile

Syntax

int FAR PASCAL edEditBeginningOfFile(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise it is zero.

Comments

edEditBeginningOfFile moves the current position to Line 1, Column 1.

edEditEndOfFile

Syntax

int FAR PASCAL edEditEndOfFile(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise it is zero.

Comments

edEditEndOfFile moves the current position to the column following the last text character at the end of the file.

edEditDownLine

Syntax

int FAR PASCAL edEditDownLine(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edEditDownLine moves the current position to the next line.

edEditUpLine

Syntax

int FAR PASCAL edEditUpLine(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edEditUpLine moves the current position to the previous line.

edEditLeft

Syntax

int FAR PASCAL edEditLeft(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edEditLeft moves the current position one column to the left. If the current position is Column 1, the current position is moved to the end of the previous line.

edEditRight

Syntax

int FAR PASCAL edEditRight(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edEditRight moves the current position one column to the right.

edEditPageUp

Syntax

int FAR PASCAL edEditPageUp(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edEditPageUp moves the current position one screenful of lines up.

edEditPageDown

Syntax

int FAR PASCAL edEditPageDown(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edEditPageDown moves the current position one screenful of lines down.

edEditWordLeft

Syntax

int FAR PASCAL edEditWordLeft(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edEditWordLeft moves the current position one word to the left.

edEditWordRight

Syntax

int FAR PASCAL edEditWordRight(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edEditWordRight moves the current position one word to the right.

edEditStartSelection

Syntax

int FAR PASCAL edEditStartSelection(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edEditStartSelection marks the beginning position of a new selection. Any previous selection is cleared.

edEditEndSelection

Syntax

int FAR PASCAL edEditEndSelection(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edEditEndSelection completes the marking of a selection started with edEditStartSelection.

edEditClearSelection

Syntax

int FAR PASCAL edEditClearSelection(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edEditClearSelection removes any selection marks.

edEditSetBookMark

Syntax

int FAR PASCAL edEditSetBookMark(HWND hWnd, int iMark)

Parameters

HWND hWnd

Identifies the WinEdit window

int iMark

Identifies the mark to be set.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edEditSetBookMark sets a mark at the current position. The caret can subsequently be moved to that position with edEditGoToBookMark.

edEditGoToBookMark

Syntax

int FAR PASCAL edEditGoToBookMark(HWND hWnd, int iMark)

Parameters

HWND hWnd

Identifies the WinEdit window.

int iMark

Identifies the mark to go to.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edEditGoToBookMark moves the current position to the mark previously set with a call to edEditSetBookMark.

edEditTab

Syntax

int FAR PASCAL edEditTab(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edEditTab inserts spaces and moves the current position to the next tab stop. If there is a selection, every line within the selection is shifted to the right one tab stop.

edEditBackTab

Syntax

int FAR PASCAL edEditBackTab(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edEditBackTab moves the current position to the previous tab stop. If there is a selection, every line within the selection is shifted to the left one tab stop.

edEditGetCurrentWord

Syntax

int FAR PASCAL edEditGetCurrentWord(HWND hWnd, LPSTR lpBuffer, int iLength)

Parameters

HWND hWnd

Identifies the WinEdit window.

LPSTR lpBuffer

A buffer for the returned word.

int iLength

The length of lpBuffer.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edEditGetCurrentWord fills lpBuffer with the word at the current position. If the caret is not on an alphanumeric character, lpBuffer is not filled.

edSearchFind

Syntax

int FAR PASCAL edSearchFind(HWND hWnd, LPSEARCHRECORD lpSearch)

Parameters

HWND hWnd

Identifies the WinEdit window.

LPSEARCHRECORD lpSearch

Identifies the search parameters to be used.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edSearchFind searches for the text identified by the lpSearchText field of lpSearch. The lpReplaceText field of lpSearch is ignored.

See Also

[SEARCHRECORD](#)

edSearchRepeat

Syntax

int FAR PASCAL edSearchRepeat(HWND hWnd, LPSEARCHRECORD lpSearch)

Parameters

HWND hWnd

Identifies the WinEdit window.

LPSEARCHRECORD lpSearch

Identifies the search parameters to be used.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edSearchRepeat conducts a search using the same search string used in the previous search. The lpSearchText and lpReplaceText fields of lpSearch are ignored.

See Also

[SEARCHRECORD](#)

edSearchChange

Syntax

int FAR PASCAL edSearchChange(HWND hWnd, LPSEARCHRECORD lpSearch)

Parameters

HWND hWnd

Identifies the WinEdit window.

LPSEARCHRECORD lpSearch

Identifies the search parameters to be used.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edSearchChange searches for the text identified by the lpSearchText field of lpSearch and replaces it with the text identified by the lpReplaceText field of lpSearch.

See Also

[SEARCHRECORD](#)

edSearchNextError

Syntax

int FAR PASCAL edSearchNextError(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edSearchNextError displays the next warning or error message on the status line.

edSearchPrevError

Syntax

int FAR PASCAL edSearchPrevError(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edSearchPrevError displays the previous warning or error message on the status line.

edSearchViewOutput

Syntax

int FAR PASCAL edSearchViewOutput(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edSearchViewOutput loads the captured output from a compilation into an MDI child window.

edRunCompile

Syntax

int FAR PASCAL edRunCompile(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edRunCompile executes the Run command. The command is set in the Run.Configure dialog box.

See Also

[edRunConfigure](#)

edRunMake

Syntax

int FAR PASCAL edRunMake(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edRunMAke executes the Make command. The command is set in the Run.Configure dialog box.

See Also

[edRunConfigure](#)

edRunRebuild

Syntax

int FAR PASCAL edRunRebuild(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edRunRebuild executes the Rebuild command. The command is set in the Run.Configure dialog box.

See Also

[edRunConfigure](#)

edRunDebug

Syntax

int FAR PASCAL edRunDebug(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edRunDebug executes the Debug command. The command is set in the Run.Configure dialog box.

See Also

[edRunConfigure](#)

edRunExecute

Syntax

int FAR PASCAL edRunExecute(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edRunExecute executes the Execute command. The command is set in the Run.Configure dialog box.

See Also

[edRunConfigure](#)

edRunCommand

Syntax

int FAR PASCAL edRunCommand(HWND hWnd, BOOL bWait, BOOL bCapture, LPSTR lpCommand)

Parameters

HWND hWnd

Identifies the WinEdit window.

BOOL bWait

If TRUE, WinEdit won't return until the process has completed.

BOOL bCapture

If TRUE, any character output from the process will be captured in a file named EDOUT. Output in Microsoft or Borland error format can be parsed and displayed with calls to edViewNextError and edViewPrevError.

LPSTR lpCommand

Identifies the command, including any command line parameters, to execute.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

Only the output from DOS character mode programs which write to stdout can be captured.

edRunConfigure

Syntax

int FAR PASCAL edRunConfigure(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edRunConfigure brings up the dialog box which allows the user to configure the Run commands.

edWindowTile

Syntax

int FAR PASCAL edWindowTile(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edWindowTile tiles all MDI child windows. If there are three or less windows, the windows will be tiled horizontally.

edWindowCascade

Syntax

int FAR PASCAL edWindowCascade(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edWindowCascade cascades the MDI child windows.

edWindowArrangelcons

Syntax

int FAR PASCAL edWindowArrangelcons(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edWindowArrangelcons orders all minimized MDI child windows.

edWindowMinimize

Syntax

int FAR PASCAL edWindowMinimize(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edWindowMinimize minimizes the active MDI child window.

edWindowMaximize

Syntax

int FAR PASCAL edWindowMaximize(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edWindowMaximize maximizes the active MDI child window.

edWindowRestore

Syntax

int FAR PASCAL edWindowRestore(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edWindowRestore restores the active MDI child window to its non-minimized, non-maximized state.

edWindowClose

Syntax

int FAR PASCAL edWindowClose(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edWindowClose closes the active MDI child window. If there are unsaved changes, the user is prompted to save the changes before closing.

edWindowCloseAll

Syntax

int FAR PASCAL edWindowCloseAll(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edWindowCloseAll closes all MDI child windows. If there are unsaved changes, the user is prompted to save the changes before closing.

edHelpIndex

Syntax

int FAR PASCAL edHelpIndex(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edHelpIndex calls WinHelp and displays the main WinEdit help index.

edHelpKeyboard

Syntax

int FAR PASCAL edHelpKeyboard(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edHelpIndex calls WinHelp and displays the 'keyboard' help topic.

edHelpCommands

Syntax

int FAR PASCAL edHelpCommands(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edHelpCommands calls WinHelp and displays the 'commands' help topic.

edHelpProcedures

Syntax

int FAR PASCAL edHelpProcedures(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edHelpProcedures calls WinHelp and displays the 'procedures' help topic.

edHelpKeyWord

Syntax

int FAR PASCAL edHelpKeyWord(HWND hWnd)

Parameters

HWND hWnd

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edHelpKeyWord retrieves the current word and uses that as a help topic for Windows API help. WinEdit looks for a Windows API help file in this order:

- The help file identified by the SDKHELP entry in WINEDIT.INI
- WIN31WH.HLP file
- QCWIN.HLP
- SDKWIN.HLP

edHelpHelp

Syntax

int FAR PASCAL edHelpHelp(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edHelpCommands calls WinHelp and displays the 'using help' help topic.

edHelpAbout

Syntax

int FAR PASCAL edHelpAbout(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edHelpAbout displays WinEdit's About dialog box.

edGetModifiedStatus

Syntax

int FAR PASCAL edGetModifiedStatus(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

TRUE if the active MDI child has been modified.

edGetLineNumber

Syntax

int FAR PASCAL edGetLineNumber(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The line number of the current position in the active MDI child window if successful, 0 if not.

edGetColumnNumber

Syntax

int FAR PASCAL edGetColumnNumber(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The column number of the current position in the active MDI child window if successful, 0 if not.

edGetSelectionState

Syntax

int FAR PASCAL edGetSelectionState(HWND hWnd, LPPOINT ptStart, LPPOINT ptEnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

LPPOINT ptStart

The x field contains the line number and the y field contains the column number of the start of the selection.

LPPOINT ptEnd

The x field contains the line number and the y field contains the column number of the end of the selection.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

edGetUndoState

Syntax

int FAR PASCAL edGetUndoState(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if any operation can be undone. Otherwise the result is zero.

edGetRedoState

Syntax

int FAR PASCAL edGetRedoState(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The result is nonzero if any operation can be redone. Otherwise the result is zero.

edGetWordWrapState

Syntax

int FAR PASCAL edGetWordWrapState(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

TRUE if word wrap is enabled, FALSE otherwise.

edGetInsertState

Syntax

int FAR PASCAL edGetInsertState(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

TRUE if Insert is on, FALSE if Overwrite is on.

edGetWindowName

Syntax

int FAR PASCAL edGetWindowName(HWND hWnd, LPSTR lpBuffer, int iSize)

Parameters

HWND hWnd

Identifies the WinEdit window

LPSTR lpBuffer

Buffer to hold the returned file name.

int iSize

Length of lpBuffer.

Return Value

The result is nonzero if the operation was successful. Otherwise the result is zero.

Comments

edGetWindowName fills lpBuffer with the fully qualified path name of the active MDI child window.

edGetChar

Syntax

int FAR PASCAL edGetChar(HWND hWnd)

Parameters

HWND hWnd

Identifies the WinEdit window.

Return Value

The character at the current position.

SEARCHRECORD

```
typedef struct tagSEARCHRECORD
{
    UINT    fComplain:    1;    /* show 'not found' etc. to user    */
    UINT    fPrompt:      1;    /* bring up search/replace dialog box */
    UINT    fMatchCase:   1;    /* exact case match only            */
    UINT    fForward:     1;    /* forward direction                 */
    UINT    fChangeAll:   1;    /* change all                         */
    UINT    fConfirm:     1;    /* ask user to confirm changes       */
    UINT    fUnused:      11;
    LPSTR   lpSearchText;    /* string to search for              */
    LPSTR   lpReplaceText;  /* string to replace found text with */
} SEARCHRECORD;

typedef SEARCHRECORD FAR *LPSEARCHRECORD;
```

The SEARCHRECORD structure holds the information used in calls to the edSearchFind, edSearchRepeat, and edSearchChange functions.

Member Description

fComplain	If this flag is set, a "Not found" message box will be shown to the user on unsuccessful searches.
fPrompt	If this flag is set, the search and/or replace information will be obtained through the use of a dialog box rather than the SEARCHRECORD information.
fMatchCase	If this flag is set, case sensitivity is turned on.
fForward	If this flag is set, the search is in a forward direction from the current position.
fChangeAll	If this flag is set, the replace operation continues until cancelled by the user or the end of file is reached.
fConfirm	If this flag is set, the user will be prompted to confirm each replacement.
fUnused	Reserved.
lpSearchText	A LPSTR to the text string to be searched for. This field must contain a valid zero terminated string unless the fPrompt flag is set.
lpReplaceText	A LPSTR to the text string to be used as a replacement. This field must contain a valid zero terminated string in calls to edSearchReplace unless the fPrompt flag is set.

See Also

[edSearchFind](#)
[edSearchRepeat](#)
[edSearchChange](#)

Configuring the Utility Menu

The Utility Menu is a custom menu created with WIL language extension commands. The menu items on the Utility Menu and the functions behind them are totally configurable. The WINEDIT.MNU file contains the menu item text and the WIL language functions behind them. The functions are made up of statements, constants, and variables, much like any programming language. To edit the Utility Menu file choose "Edit Utility Menu" from the Utility Menu.

In the WINEDIT.MNU file, the menu item text that appears below the Utility Menu begins in Column one of the text file. Commands for the menu item are at least 8 spaces to the right below the menu text. In the example below, "File Open with prompt" is the text that will appear on the Utility Menu and the "WFileOpen("")" command will be executed if the menu item is selected.

```
File Open with prompt      ; Open a document file via dialog box
      WFileOpen("")
```

wFileOpen(filename)

Comments

wFileOpen creates a new MDI child window and reads an existing file into the window. To open a file without a prompt, pass a valid filename to wFileOpen. If the FileName parameter is "", the File Open dialog box will be appear prompting for a filename to open.

To create a pop-out submenu, add one space before the menu title for every item of the main menu. For example, "Accessories" will appear on the Utility Menu with "Appointment Scheduling" and "Calculator" appearing as pop out menu selections.

```
Accessories
  Appointment Scheduling
      run("Calendar.exe", "")
  C&alculator
      run("calc.exe", "")
```

Adding an ampersand before any letter in the title causes that letter to be displayed underlined. Such underlined letters are recognized by Windows as menu hot keys accessed through an ALT+letter key combination.

See Also:

[Utility Menu](#)

[WIL Commands](#)

WIL Commands

In addition to the following WinEdit WIL commands you can use the WIL (Windows Interface Language) commands when configuring the Utility Menu file (WINEDIT.MNU). Look to the WIL.HLP file for more information on the Windows Interface Language and its available commands.

wAddButton
wCallMacro
wDelButton
wChange
wEdBackspace
wEdCopy
wEdPaste
wEdCopyLine
wEdCut
wEdCutLine
wEdDelete
wEdGoToLine
wEdGoToCol
wEdHome
wEdEnd
wEdTopOfFile
wEdEndOfFile
wEdUpLine
wEdDownLine
wEdLeft
wEdRight
wEdPageUp
wEdPageDown
wEdWordLeft
wEdWordRight
wEdStartSel
wEdEndSel
wEdTab
wEdBackTab
wEdGetWord
wEdSelectAll
wEdInsString
wEdSetColBlk
wEdWrap
wEdToggleIns
wEdRedo
wEdUndo
wFileList
wFileNew
wFileOpen
wFileMerge
wFileSave
wFileSaveAs
wFilePrint
wFilePgSetup
wPrinSetup
wFileExit
wFind

wGetChar
wGetFileName
wGetIns
wGetSelState
wGetRedo
wGetUndo
wGetWrap
wGetColNo
wGetLineNo
wGetModified
wHelpAbout
wHelpCmds
wHelpKeybrd
wHelpKeyWord
wHelpHelp
wHelpIndex
wNextError
wPrevError
wRepeat
wRunConfig
wRunCommand
wRunCompile
wRunDebug
wRunExecute
wRunMake
wRunRebuild
wSetPrefs
wViewOutput
wWinArricons
wWinCascade
wWinClose
wWinCloseAll
wWinMaximize
wWinMinimize
wWinRestore
wWinTile

wAddButton

wAddButton(icon, command, position)

Comments

wAddButton adds a new button to the control bar.

Icon is the constant identifier indicating which icon to use and can be one of the following values:

@openicon	@makeicon
@saveicon	@rebuildicon
@listicon	@debugicon
@printicon	@executeicon
@findicon	@cuticon
@repeaticon	@copyicon
@changeicon	@pasteicon
@nexticon	@undoicon
@previcon	@redoicon
@compileicon	@helpkeyicon

The command parameter is the constant identifier indicating which command to execute when the user clicks the icon. The command parameter can be one of the following values:

@wfilenew	@wfileopen
@wfilemerge	@wfilelist
@wfilesave	@wfilesaveas
@wfileprint	@wfilepgstup
@wprinsetup	@wsetprefs
@wfileexit	@wedundo
@wedredo	@wedcut
@wedcopy	@wedpaste
@weddelete	@wedtoggleins
@wedwrap	@wedsetcolbk
@wedinsstrng	@wedbackspce
@wedselctall	@wedcopyline
@wedcutline	@wedgotoline
@wedgotocol	@wedhome
@wedend	@wetopoffile
@weendoffile	@wedupline
@weddownline	@wedleft
@wedright	@wedpageup
@wedpagedown	@wedwordleft
@wewordright	@wedstartsel
@wedendsel	@wedclearsel
@wedtab	@wedbacktab
@wedgetword	@wfind
@wrepeat	@wchange
@wnexterror	@wpreverror
@wviewoutput	@wruncompile
@wrunmake	@wrunrebuild
@wrundebug	@wrunexecute
@wruncommand	@wrunconfig
@wwintile	@wwincascade
@wwnarrIcons	@wwnminimize
@wwnmaximize	@wwinrestore

@wwinclose	@wwncloseall
@whelpindex	@whelpkeybrd
@whelpcmds	@whlpkeyword
@whelphelp	@whelpabout
@wgtmodified	@wgetlineno
@wgetcolno	@wgtsselstate
@wgetundo	@wgetredo
@wgetwrap	@wgetins
@wgtfilename	@wgetchar
@waddbutton	@wdelbutton
@wcall1	@wcall2
@wcall3	@wcall4
@wcall5	@wcall6
@wcall7	@wcall8
@wcall9	@wcall10
@wcall11	@wcall12
@wcall13	@wcall14
@wcall15	@wcall16
@wcall17	@wcall18
@wcall19	@wcall20

The position parameter is the 0-based position on the control bar for the button to be added. (the first button is position 0 and the last is position 16)

Example:

```
wAddButton(@debugicon, @wcall1, 12)
```

The above example will add the debug icon to the control bar at position 12. The "@wcall1" parameter will run the macro commands in the WINEDIT.WBT file for label number 1. Look to the WINEDIT.WBT file for more information.

See Also:

[wDelButton](#)

wCallMacro

wCallMacro(label)

Comments

wCallMacro calls a user-defined macro defined in the WINEDIT.WBT file. Label is a string which is used as a label in the WINEDIT.WBT file. In order to use the wCallMacro function with control bar icons, the label for each macro must be a number from 1 through 20 (or @wcall1-@wcall20) - see [wAddButton](#) for an example.

Example:

```
wCallMacro (1)
```

The above command calls the macro defined for label number 1 in the WINEDIT.WBT file. See the WINEDIT.WBT file for more information.

wDelButton

wDelButton(position,command)

Comments

wDelButton deletes the button at the indicated position. Command is the command constant as described for the wAddButton command (see [wAddButton](#)). The default WinEdit control bar has the following commands:

Position	Command
-----	-----
0	@WFileOpen
1	@WFileSave
2	@WFileList
3	@WFilePrint
4	@WEdUndo
5	@WEdRedo
6	@WEdCut
7	@WEdCopy
8	@WEdPaste
9	@WFind
10	@WRepeat
11	@WChange
12	@WRunCompile
13	@WRunMake
14	@WRunRebuild
15	@WRunDebug
16	@WRunExecute
17	@WPrevError
18	@WNextError
19	@WHlpKeyword

See Also:

[wAddButton](#)

wChange

wChange(SearchText, ReplaceText, Forward, MatchCase, ChangeAll)

Comments

wChange searches for the specified SearchText and replaces it with ReplaceText. If ChangeAll is equal to 1 (True), then the search and replace will continue to the end of the file.

Example:

```
wChange("Blue", "Red", 1, 0, 1)
```

The above command will start at the cursor position and search through to the end of the file, replacing text string "Blue" with "Red". The MatchCase argument is set to False or 0, so the search string "Blue" will be changed to "Red" regardless of the case of the word blue in the document.

wEdBackspace

wEdBackspace()

Comments

wEdBackSpace deletes the character to the left of the current position. This command is the equivalent of pressing the backspace character on the keyboard.

Example:

```
wEdBackSpace ()  
wEdHome ()
```

The above example deletes the character to the left of the cursor and moves the cursor to the beginning of the line.

wEdCopy

wEdCopy()

Comments

wEdCopy copies the selected text to the Windows clipboard.

Example:

```
wEdStartSel ()  
wEdWordLeft ()  
wEdEndSel ()  
wEdCopy ()
```

The above commands will select the word to the left of the cursor and copy it to the Windows clipboard.

wEdPaste

wEdPaste()

Comments

wEdPaste pastes text from the clipboard into the active WinEdit document window.

Example:

```
wEdSelectAll()  
wEdCopy()  
wFileNew()  
wEdPaste()
```

The above commands will copy the contents of the active document window and paste the contents of the window into a new document window.

wEdCopyLine

wEdCopyLine()

Comments

wEdCopyLine copies the current line to the clipboard if there is no selection. If there is a selection, wEdCopyLine calls wEdCopy and copies the selected text to the clipboard.

Example:

```
wEdCopyLine()  
wEdDownLine()  
wEdPaste()
```

The above example copies the line of text where the cursor resides, moves down a line, and pastes the line of text from the clipboard.

wEdCut

wEdCut()

Comments

wEdCut cuts the current selection to the clipboard. The text cut to the clipboard can be later inserted into a document with the wEdPaste command. This command requires that text is selected. If nothing is selected, the wEdCut() command will return the following message:

Nothing selected to cut.

See Also:

[wEdDelete](#)

[wEdPaste](#)

wEdCutLine

wEdCutLine()

Comments

wEdCutLine cuts the current line to the clipboard if there is no selection. If text is selected, then wEdCutLine calls wEdCut and cuts the selected text to the clipboard.

Example:

```
wEdCutLine()  
wEdGoToLine(4)  
wEdPaste()
```

The above example cuts the contents of the current line to the clipboard and pastes the line on line 4 of the active document.

wEdDelete

wEdDelete()

Comments

wEdDelete deletes either the current selection or, if there is no selection, the character following the current position without copying the text to the clipboard. This command is the equivalent of pressing the Del or Delete character on the keyboard.

Example:

```
wEdDelete ()  
wEdHome ()
```

The above example deletes the character to the right of the cursor and moves the cursor to the beginning of the line.

See Also:

[wEdCut](#)

wEdGoToLine

wEdGoToLine(lineno)

Comments

wEdGoToLine moves the current position to the line number identified by the lineno parameter. If the line number is greater than the last line in the file, the current position is moved to the last line in the file.

Example:

```
wEdGoToLine(6)
```

The above command will move the cursor to line 6 in the document file while maintaining the current column position. So if your cursor is positioned on Line 13, Col 21, the cursor position will be Line 6, Col 21 after the above command is executed.

See Also:

[wEdGoToCol](#)

wEdGoToCol

wEdGoToCol(colno)

Comments

wEdGoToCol moves the current cursor position to the column identified by the colno parameter.

Example:

```
wEdGoToCol (10)
```

The above command will move the cursor to column 10 in the document file while maintaining the current line position. So if your cursor is positioned on Line 13, Col 21, the cursor position will be Line 13, Col 10 after the above command is executed.

See Also:

[wEdGoToLine](#)

wEdHome

wEdHome()

Comments

wEdHome moves the current cursor position to Column 1 (the beginning of the line).

Example:

```
wEdHome ()  
wEdPaste ()
```

The above commands will move the cursor to the beginning of the line and paste in the contents of the clipboard.

wEdEnd

wEdEnd()

Comments

wEdEnd moves the cursor position to the column following the last text or space character in the current line.

Example:

```
wEdEnd()  
wEdInsString("Hello")
```

The above commands will insert the text Hello at the end of the current line.

wEdTopOfFile

wEdTopOfFile()

Comments

wEdTopOfFile moves the cursor position to Line 1, Column 1 (the equivalent of pressing CTRL+Home).

Example:

```
wEdTopOfFile()  
wEdInsString("Top of File")
```

The above commands will insert the text "Top of File" at the beginning of the document window (Line 1 Column 1).

wEdEndOfFile

wEdEndOfFile()

Comments

wEdEndOfFile moves the cursor position to the column following the last text character on the last line of the file (the equivalent of pressing CTRL+End).

Example:

```
wEdEndOfFile()  
wEdInsString("End of File")
```

The above commands will insert the text "End of File" after the last text in the document window.

wEdUpLine

wEdUpLine()

Comments

wEdUpLine moves the current cursor position to the previous line (moves to the line above the current line).

Example:

```
wEdUpLine ()  
wEdHome ()
```

The above commands will move the cursor position to the beginning of the previous line.

wEdDownLine

wEdDownLine()

Comments

wEdDownLine moves the current position to the next line (moves to the line below the current line).

Example:

```
wEdDownLine()  
wEdEnd()
```

The above commands will move the cursor position to the end of the next line.

wEdLeft

wEdLeft()

Comments

wEdLeft moves the current position one column to the left. If the current position is Column 1, the current position is moved to the end of the previous line.

Example:

```
wEdLeft ()  
wEdTab ()
```

The above commands will move the cursor position one position to the left and insert a tab (the number of spaces for the tab character is set in File Preferences).

wEdRight

wEdRight()

Comments

wEdRight moves the current position one column to the right.

Example:

```
wEdRight()  
wEdTab()
```

The above commands will move the cursor position one position to the right and insert a tab (the number of spaces for the tab character is set in File Preferences).

wEdPageUp

wEdPageUp()

Comments

wEdPageUp moves the current position up one screenful of text (equivalent of pressing PgUp on the keyboard).

wEdPageDown

wEdPageDown()

Comments

wEdPageDown moves the current position down one screenful of text (equivalent of pressing PgDn on the keyboard).

wEdWordLeft

wEdWordLeft()

Comments

wEdWordLeft moves the cursor position one word to the left (the cursor will be positioned just before the word to the left of the current cursor position).

wEdWordRight

wEdWordRight()

Comments

wEdWordRight moves the current position one word to the right (the cursor will be positioned just before the word to the right of the current cursor position).

wEdStartSel

wEdStartSel()

Comments

wEdStartSel marks the beginning position of a new selection. Any previous selection is cleared.

Example:

```
wEdStartSel ()  
wEdWordRight ()  
wEdEndSel ()  
wEdCopy ()
```

The above commands will copy the word to the right of the cursor position into the Windows clipboard (use the Edit Paste menu command or wEdPaste() to retrieve the text).

wEdEndSel

wEdEndSel()

Comments

wEditEndSel completes the marking of a selection started with wEdStartSel.

Example:

```
wEdStartSel ()  
wEdWordRight ()  
wEdWordRight ()  
wEdEndSel ()  
wEdCopy ()  
wEdHome ()  
wEdPaste ()
```

The above commands will copy the two words to the right of the cursor position and paste the two words at the beginning of the current line.

wEdTab

wEdTab()

Comments

wEdTab inserts a number of spaces and moves the current position to the next tab stop. If more than one line is selected, every line within the selection is shifted to the right one tab stop. The amount of spaces that is inserted is set in the Preferences dialog (choose Preferences from the File menu).

wEdBackTab

wEdBackTab()

Comments

wEdBackTab moves the current position to the previous tab stop. If there is a selection, every line within the selection is shifted to the left one tab stop. The amount of spaces that the text is shifted is a settings in the Preferences dialog (choose Preferences from the File menu).

wEdGetWord

wEdGetWord()

Comments

wEdGetWord returns the word at the current cursor position. If the cursor is not on an alphanumeric character, an empty string is returned.

Example:

```
A=wEdGetWord()  
Message("Title",A) ; WIL Command, see WIL.HLP
```

The above commands get the word where the insertion point is positioned and assign the text to the variable "A". The Message command is used to display the contents of the A variable in a message box. The "Message" command is a WIL (Windows Interface Language) command. Look to the WIL.HLP file for more information on the WIL commands.

wEdSelectAll

wEdSelectAll()

Comments

wEdSelectAll selects all the text in the active document window. The insertion position is moved to the end of the file.

Example:

```
wEdSelectAll ()  
wEdCopy ()  
wFileNew ()  
wEdPaste ()
```

The above commands will copy the contents of the active document window and paste the contents of the window into a new document window.

wEdInsString

wEdInsString(string)

Comments

wEdInsString inserts string at the current position.

Example:

```
A=wEdGetWord()  
wEdDownLine()  
wEdGoToCol(1)  
wEdInsString(A)
```

The above commands get the word where the insertion point is positioned and assign the text to the variable "A". The remaining commands inserts the contents of the A variable at the beginning of the next line.

wEdSetColBlk

wEdSetColBlk()

Comments

wEdSetColBlk enables column block marking for the next block operation. WinEdit automatically returns to stream block marking after the next block operation.

Example:

```
wEdStartSel ()
wEdSetColBlk ()
wEdGoToCol (10)
wEdDownLine ()
wEdEndSel ()
wEdCopy ()
wEdDownLine ()
wEdGoToCol (1)
wEdPaste ()
```

The first five lines above will block select 10 characters to the right of the insertion point on the current line and the line below. Once marked, the text is copied to the clipboard and inserted and at the beginning of the following line.

wEdWrap

wEdWrap()

Comments

wEdWrap toggles the word wrap state on or off. If Word Wrap is selected under the Edit menu (turned "on"), then the wEdWrap() command will toggle word wrap "off".

See Also:

[wGetWrap](#)

wEdToggleIns

wEdToggleIns()

Comments

wEdToggleIns toggles the insert state between Insert and Overtyping modes (INS or OVR indicates the insert state on the status bar). If Insert Mode is selected under the Edit menu (turned "on"), then the wEdToggleIns() command will toggle to OverType mode.

See Also:

[wGetIns](#)

wEdRedo

wEdRedo()

Comments

Equivalent of selecting Redo from the Edit menu. The wEdRedo() command allows you to reverse any Undo command.

See Also:

[wGetRedo](#)

wEdUndo

wEdUndo()

Comments

Allows you to "undo" the most recent editing action.

See Also:

[wGetUndo](#)

wFileList

wFileList()

Comments

wFileList brings up the Reopen File dialog box which lists the last 20 documents opened (same as pressing F4 or choosing Previous Files from the File menu)

wFileNew

wFileNew()

Comments

wFileNew creates a new MDI child window.

Example:

```
wEdSelectAll()  
wEdCopy()  
wFileNew()  
wEdPaste()
```

The above commands will copy the contents of the active document window and paste the contents of the window into a new document window.

wFileOpen

wFileOpen(filename)

Comments

wFileOpen creates a new MDI child window and reads an existing file into the window. To open a file without prompting, pass a valid file name to wFileOpen. If the FileName parameter is "", the File Open dialog box will appear prompting the user for a filename.

Example:

```
wFileOpen("")
```

The above command will prompt the user for a filename to open. To open a file directly without prompting, use the following syntax:

```
wFileOpen("FILENAME.TXT")
```

wFileMerge

wFileMerge(filename)

Comments

wFileMerge reads an existing file into the active MDI child window. To merge a file without prompting, pass a valid file name to wFileMerge in the FileName parameter. If FileName is "", the File Merge dialog box will be used to obtain a file name from the user.

Example:

```
wFileMerge("")
```

The above command will prompt the user for a filename to merge. To merge in a file directly without prompting, use the following syntax:

```
wFileMerge("FILENAME.TXT")
```

The indicated file is merged at the insertion position in the active document window.

wFileSave

wFileSave()

Comments

wFileSave saves the file in the currently active MDI child window without prompting (same as selecting Save from the File menu).

wFileSaveAs

wFileSaveAs(filename)

Comments

wFileSaveAs saves the file in the currently active MDI child window to a new filename.

Example:

```
wFileSaveAs("")
```

The above command will prompt the user for a filename. To save the file directly to new file name without prompting, use the following syntax:

```
wFileSaveAs("FILENAME.TXT")
```

wFilePrint

wFilePrint()

Comments

wFilePrint prints the text in the currently active MDI child window (same as choose Print from the File menu).

wFilePgSetup

wFilePgSetup()

Comments

wFilePgSetup brings up the Page Setup dialog box (same as choosing Page Setup from the File menu).

wPrinSetup

wPrinSetup()

Comments

wPrinSetup brings up a dialog box listing all installed printers (same as selecting Printer Setup from the File menu). The user can choose a printer from the list and WinEdit will use the selected driver for all print jobs. The user can also access the printer driver setup dialog by choosing the Setup button.

wFileExit

wFileExit()

Comments

Command to exit WinEdit. If there are any unsaved files, the user will be prompted to save before closing. The user can cancel the exit operation at that point. If there are no unsaved files, the exit is automatic (no chance to cancel the exit).

wFind

wFind(SearchText,Forward,MatchCase)

Comments

wFind searches for the text identified by SearchText parameter. If Forward is TRUE, the search direction is forward. If MatchCase is TRUE, then the search is case sensitive.

Example:

```
wFind("Blue",1,1)
```

The above example searches forward through the document window for the word Blue.

wGetChar

wGetChar()

Return Value

Returns the character to the right of the insertion point.

Example:

```
a=wGetChar()  
wEdInsString(a)
```

This example gets the character to the right of the insertion point and inserts the character into the document window.

wGetFileName

wGetFileName()

Comments

wGetFileName returns a string with the fully qualified path name of the active MDI child window.

Example:

```
a=wGetFileName()  
wEdInsString(a)
```

This example gets the filename for the active document window and inserts the filename (with the path information) at the insertion point.

wGetIns

wGetIns()

Return Value

Returns TRUE (1) if Insert is on, FALSE (0) if Overtyping is on.

Example:

```
a=wGetIns()  
If a == 0 Then Message ("Title", "Overtyping is on")  
If a == 1 Then Message ("Title", "Insert Mode is on")
```

The above commands assign the return value of wGetIns() to the "a" variable and then test for whether "a" is True or False. The If command used above to evaluate the "a" variable is a WIL (Windows Interface Language) command. Look to the WIL.HLP file for more information on the WIL commands.

See Also:

[wEdToggleIns](#)

wGetSelState

wGetSelState()

Return Value

The result is TRUE if there is a selection, otherwise the function returns zero.

Example:

```
a=wGetSelState()  
If a == 1 Then wEdCopy()
```

This example checks whether there is a selection, and if True copies the selection to the clipboard.

wGetRedo

wGetRedo()

Return Value

The result is TRUE (1) if any operation can be redone. Otherwise wGetRedo returns zero.

Example:

```
a=wGetRedo()  
If a == 1 Then wEdRedo()
```

The above example checks whether the last edit can be redone and if the return value is TRUE, the edit is redone ("wEdRedo()") is the same as choosing Redo from the Edit menu).

See Also:

[wEdRedo](#)

wGetUndo

wGetUndo()

Return Value

The result is TRUE (1) if any operation can be undone. Otherwise wGetUndo returns zero. ("wEdUndo()") is the same as choosing Undo from the Edit menu).

Example:

```
a=wGetUndo()  
If a == 1 Then wEdUndo()
```

The above example checks whether the last edit can be undone and if the return value is TRUE, the edit if undone.

See Also:

[wEdUndo](#)

wGetWrap

wGetWrap()

Return Value

The result is TRUE if word wrap is enabled, FALSE otherwise.

Example:

```
a=wGetWrap()  
If a == 0 Then Message ("Title", "Word Wrap is off")  
If a == 1 Then Message ("Title", "Word Wrap is on")
```

The above commands assign the return value of wGetWrap() to the "a" variable and then test for whether "a" is True or False. The If command used above to evaluate the "a" variable is a WIL (Windows Interface Language) command. Look to the WIL.HLP file for more information on the WIL commands.

See Also:

[wEdWrap](#)

wGetColNo

wGetColNo()

Return Value

Returns the column number position for the insertion position in the active MDI child window. wGetColNo returns 0 if unsuccessful.

Example:

```
a=wGetColNo()  
Message("Column Number", a)
```

The above commands get the column number for the insertion point and post the results in a message box. Look to the WIL.HLP file for more information on WIL commands such as the Message command.

wGetLineNo

wGetLineNo()

Return Value

Returns the line number position for the insertion position in the active MDI child window. wGetLineNo returns 0 if unsuccessful.

Example:

```
a=wGetLineNo()  
Message("Line Number", a)
```

The above commands get the line number for the insertion point and post the results in a message box. Look to the WIL.HLP file for more information on WIL commands such as the Message command.

wGetModified

wGetModified()

Return Value

TRUE if the active MDI child has been modified.

Example:

```
a=wGetModified()  
If a == 1 Then Message ("Mod", "Text has been modified")
```

The above example will post a message if the text in the document window has been modified.

wHelpAbout

wHelpAbout()

Comments

wHelpAbout displays WinEdit's About dialog box with version number and copyright information.

wHelpCmds

wHelpCmds()

Comments

wHelpCmds calls up the WinEdit Help file and displays the Menu Commands help topic.

wHelpKeybrd()

wHelpKeybrd

Comments

wHelpKeybrd calls up the WinEdit Help file and displays the Keyboard and Mouse Commands help topic.

wHelpKeyWord

wHelpKeyWord()

Comments

wHelpKeyWord retrieves the current word and uses that as a help topic for Windows API help.

wHelpHelp

wHelpHelp()

Comments

wHelpHelp calls WinHelp and displays the 'How to Use Help' topic.

wHelpIndex

wHelpIndex()

Comments

wHelpIndex calls WinHelp and displays the main WinEdit help index.

wNextError

wNextError()

Comments

wNextError displays the next warning or error message on the status line.

wPrevError

wPrevError()

Comments

wPrevError displays the previous warning or error message on the status line.

wRepeat

wRepeat()

Comments

wRepeat conducts a search using the same search string used in the previous search.

Example:

```
wFind("Blue",1,1)
PlayWaveForm("tada.wav", 0)
wRepeat()
```

This example searches forward for the word Blue, plays the TADA.WAV file and then repeats the wFind statement. The PlayWaveForm command used above is a WIL (Windows Interface Language) command. Look to the WIL.HLP file for more information on the WIL commands.

wRunConfig

wRunConfig()

Comments

wRunConfig brings up the Project Management dialog box which allows the user to configure the different run and compile commands.

wRunCommand

wRunCommand(Command,Wait,Capture)

Parameters

Command

Identifies the command, including any command line parameters, to execute.

Wait

If set to TRUE, WinEdit won't return until the process has completed.

Capture

If set to TRUE, any character output from the process will be captured in a file named EDOUT. Output in the Microsoft or Borland error format can be parsed and displayed with calls to wNextError and wPrevError.

wRunCompile

wRunCompile()

Comments

wRunCompile executes the Compile command syntax entered in the Project Management dialog box (choose Configure... from the Project menu to indicate the Compile syntax).

wRunDebug

wRunDebug()

Comments

wRunDebug executes the Debug command syntax entered in the Project Management dialog box (choose Configure... from the Project menu to indicate the Debug syntax).

wRunExecute

wRunExecute()

Comments

wRunExecute executes the Execute command syntax entered in the Project Management dialog box (choose Configure... from the Project menu to indicate the Execute syntax).

wRunMake

wRunMake()

Comments

wRunMake executes the Make command syntax entered in the Project Management dialog box (choose Configure... from the Project menu to indicate the Make syntax).

wRunRebuild

wRunRebuild()

Comments

wRunRebuild executes the Rebuild command syntax entered in the Project Management dialog box (choose Configure... from the Project menu to indicate the Rebuild syntax).

wSetPrefs

wSetPrefs()

Comments

wSetPrefs() displays the preferences dialog to allow the user to set the screen font, tab size, and other configuration options. The results are stored in WINEDIT.INI and used in future editing sessions.

wViewOutput

wViewOutput()

Comments

wViewOutput() loads the captured output from a compilation into an MDI child window (only the output from DOS character mode programs which write to stdout can be captured).

wWinArrIcons

wWinArrIcons()

Comments

wWinArrIcons rearranges all minimized MDI child windows icons along the bottom of the WinEdit application window.

Example:

```
wFileOpen("accel.rc")  
wWinMinimize()  
wFileNew()  
wWinMinimize()  
wWinArrIcons()
```

The above example opens the ACCEL.RC file and a new document window, minimizes them both and then arranges the icons left to right along the bottom of the WinEdit application window.

wWinCascade

wWinCascade()

Comments

wWinCascade cascades all MDI child windows (arranges all of the open windows in a stack).

wWinClose

wWinClose()

Comments

wWinClose closes the active MDI child window. If there are unsaved changes, the user is prompted to save the changes before the file is closed.

wWinCloseAll

wWinCloseAll()

Comments

wWinCloseAll closes all MDI child windows. If there are unsaved changes, the user is prompted to save the changes to each file before the file is closed.

wWinMaximize

wWinMaximize()

Comments

wWinMaximize maximizes the active MDI child window.

Example:

```
wFileNew()  
wWinMaximize()
```

This example opens a new document window and maximizes the window.

wWinMinimize

wWinMinimize()

Comments

wWinMinimize minimizes the active MDI child window to an icon at the bottom of the WinEdit application window.

Example:

```
wFileOpen("accel.rc")  
wWinMinimize()
```

This example opens the ACCEL.RC file and minimizes the window to an icon.

wWinRestore

wWinRestore()

Comments

wWinRestore restores the active MDI child window to its non-minimized, non-maximized state.

wWinTile

wWinTile()

Comments

wWinTile tiles all MDI child windows. If there are three or less windows, the windows will be tiled horizontally left to right.

