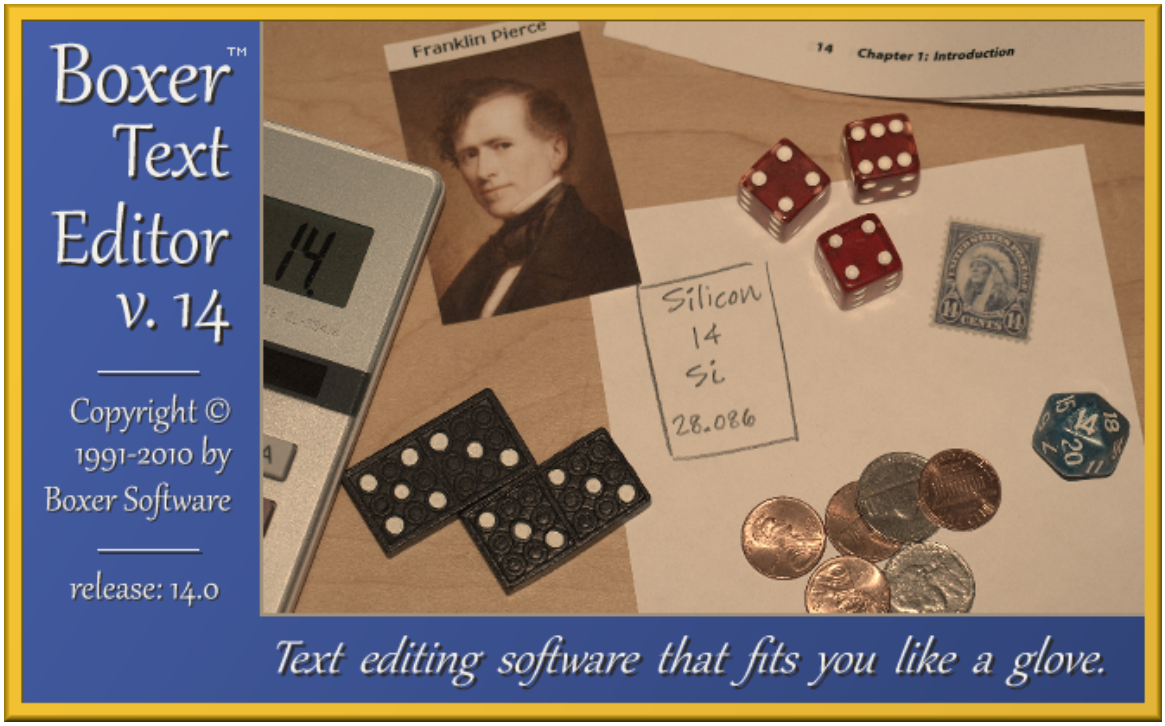
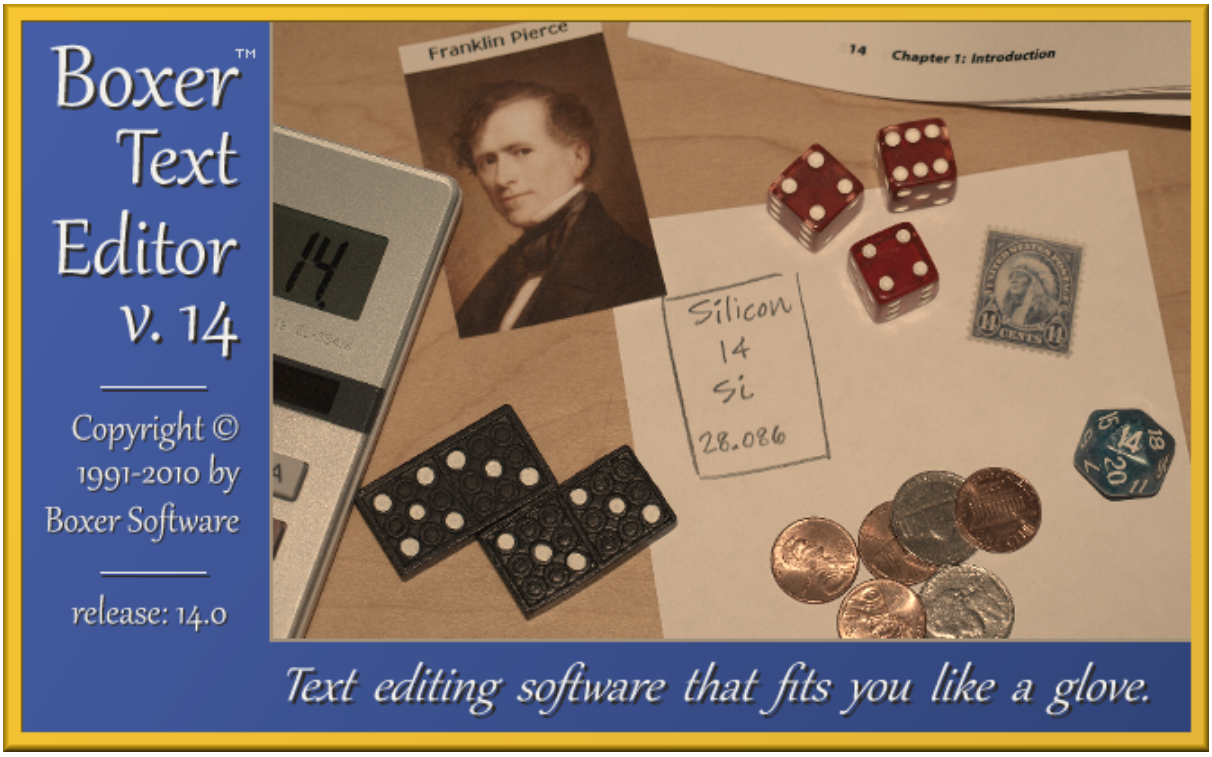
Table of Contents

1 Contents



For more information on Boxer please follow the links below:

2 Introduction



Thank you for selecting Boxer for your text editing needs. We've filled Boxer with powerful text processing capabilities that will make your editing tasks go faster and more smoothly. But we've also kept Boxer intuitive and easy-to-use so it can be used by beginners and experts, writers and programmers, high school students and scientists.

During the seventeen years we've been designing and selling our award-winning text editors, we've come to understand intimately the needs of people who edit text. We believe these are: speedy operation, a courteous and intuitive user interface, powerful editing commands, extensive configurability, prompt technical support and impeccable customer service. We've built Boxer--and our company--around all of these important principles.

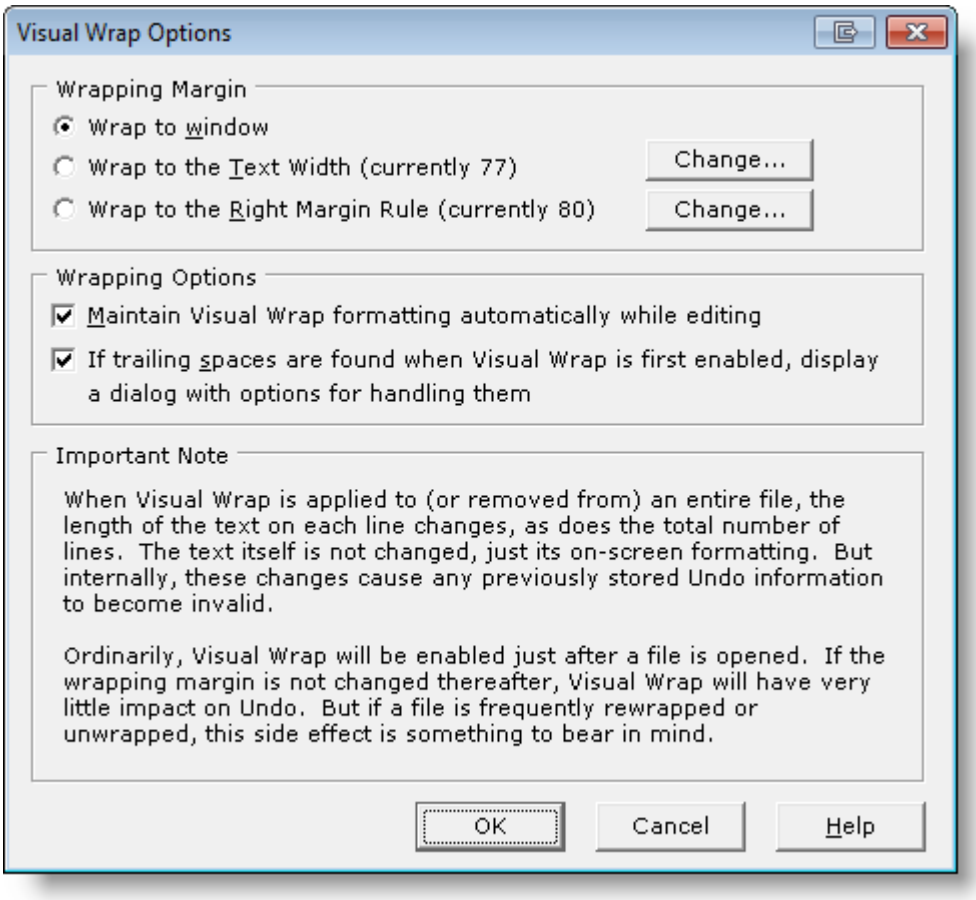
Thank you for your support!

3 New Features in Boxer 14

Version 14 adds some great new features to Boxer. The most important new features are listed below. Visit our website for a more comprehensive list of the changes.

- **Visual Wrap**

Visual Wrap is a passive display mode that allows text to be automatically wrapped to the window width without introducing hard line enders into the file. This feature is useful when editing files with very long lines which would otherwise extend off-screen to the right, out of view. The Visual Wrap Options dialog provides access to options related to the operation of Visual Wrap. Wrapping can be set to occur at the window width, the Text Width, or at the Right Margin Rule. By default, Visual Wrap is maintained when edits are made, although this can be optionally disabled. An option is also provided for dealing with trailing spaces when Visual Wrap is first applied.

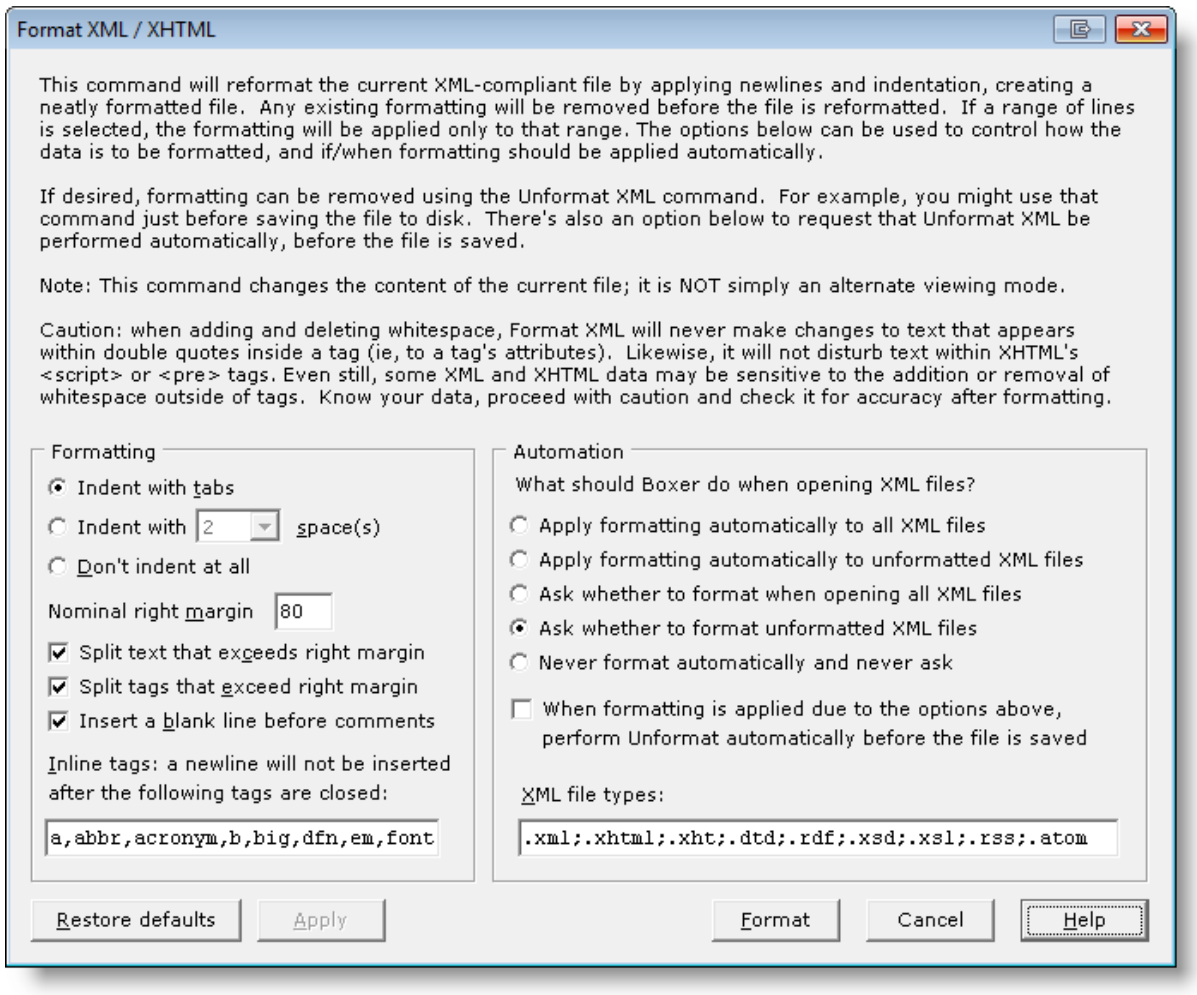


In Visual Wrap mode, the Line counter in the status bar switches to a Paragraph counter, since a one-to-one relationship between screen lines and physical lines no longer exists. Lines with a soft line ender are denoted by a trailing space character, and a single left chevron (<) is displayed if Visible Spaces mode is active. Adding a trailing space to a line converts a hard line ender to a soft line ender; removing the trailing space converts a soft line ender to a hard line ender.

Additional commands have also been added in support of Visual Wrap. The Soften Line Enders command can be applied to a text file which has unwanted line enders, thereby making the text in the file "flowable" and eligible for Visual Wrap. The Harden Line Enders command does the opposite: it converts soft line enders to hard line enders, effectively "locking" the current on-screen formatting into the file. A new Go to Paragraph command has been added to allow a jump to specified paragraph number.

- **XML Formatting**

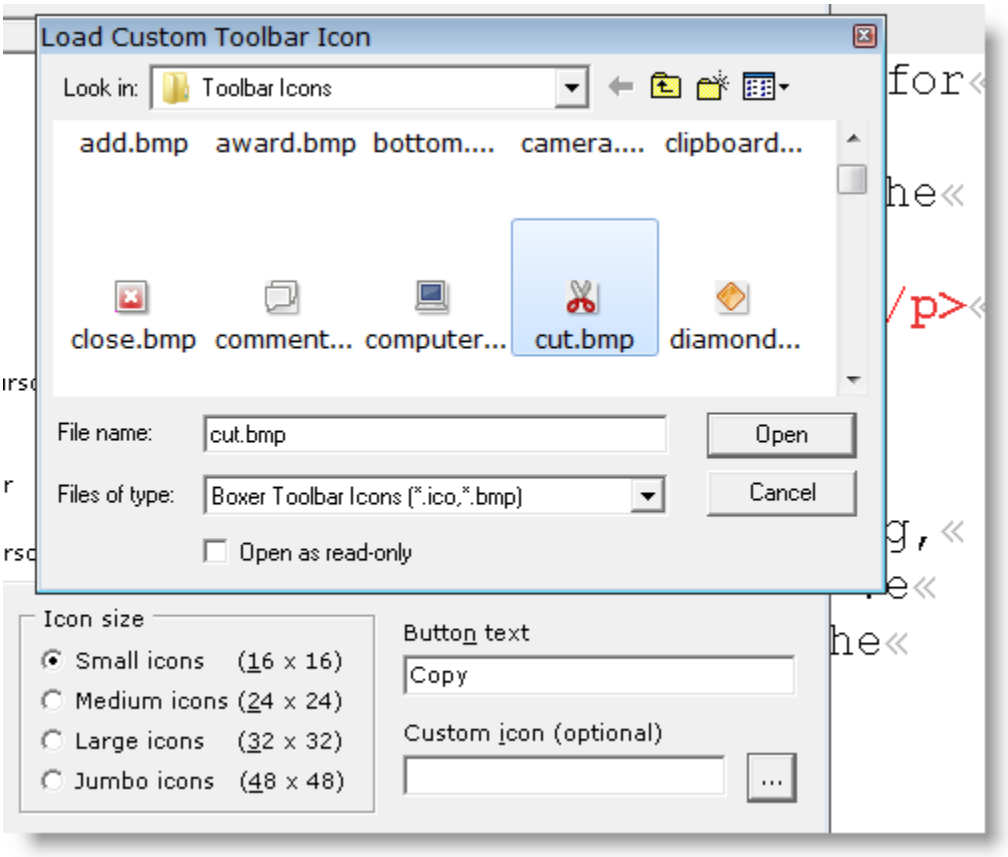
Two new commands have been added to the Tools menu: Format XML and Unformat XML. These commands can be used to apply and remove formatting, respectively, from XHTML and other XML-compliant text files. If you've ever opened an XML file and seen one long, flowing line extend off the right edge of the screen, you'll appreciate this new feature.



Format XML adds newlines and indentation intelligently to any XML-compliant file to create a neatly formatted document. Options are provided to control the amount of indent, whether to indent with spaces or tabs, when and whether text and tags should be wrapped, when to suppress newlines, and other formatting nuances. If a range of lines is selected, formatting or unformatting will be performed only on the selected lines. Upon completion, a statistics dialog provides information to assist in locating unmated or unclosed tags, or other data inconsistencies. A set of automation options provides precise control over how/if/when XML files should be auto-formatted as they are opened. The Unformat XML command can be used to convert a formatted document to "flat" form, in case this is required for processing, to reduce file size, or for other reasons.

- **User-defined Toolbar Icons**

Using the Configure Toolbar dialog, it's now possible to assign new icons to any of the toolbar buttons. New icons can be assigned to change the look of Boxer's toolbar, or to provide distinct icon images for commands that would otherwise use the same icons (Macros and User Lists are two such examples). User-defined image files can be either icon (.ICO) or bitmap (.BMP) files. Two new icon sizes are also available: in addition to the existing 16x16 and 32x32 options, 24x24 and 48x48 scaling options have been added. A collection of 64 sample images is provided with the release, and users are free to use icons from others sources as well, or to create their own.

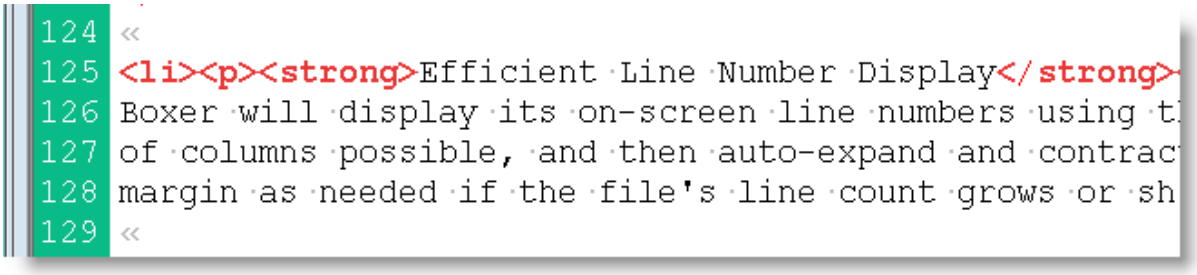


- **Windows 7 Compatibility**

Boxer 14 has been tested and tuned to run under Windows 7, Microsoft's newest operating system. Boxer runs under both the 32-bit and 64-bit versions of Windows 7.

- **Efficient Line Number Display**

Boxer will display its on-screen line numbers using the minimum number of columns possible, expanding and contracting the line number margin automatically if the file's line count grows or shrinks.

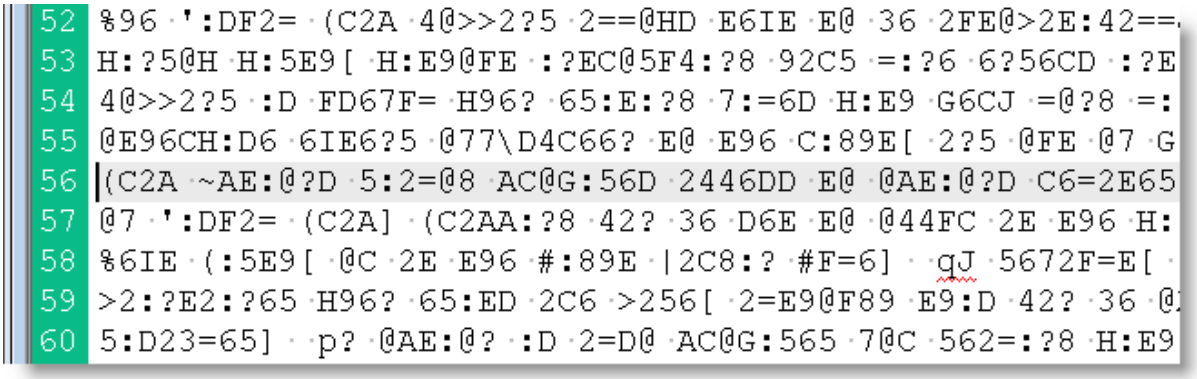


- **Move Line Up/Down**

The Move Line Up and Move Line Down commands can be used to easily move items up and down in an ordered list. These new commands appear near the bottom of the Edit menu. Though simple in concept, you'll be surprised how much these commands can speed common editing operations. (The Swap Lines command, which was nearly identical in function to the new Move Line Down command, has been removed from the menu, but remains available for use by shortcut key, or within macros.)

- **Text Obfuscation**

ROT5, ROT13, ROT18 and ROT47 conversion commands have been added to the Block| Convert Other submenu. These commands provide a simple, reversible text obfuscation scheme for use in non-critical encryption situations.



Line Spacing: this new command permits a selected range of lines, or the whole file, to be converted to single-, double- or triple-spaced format.

Support for handling ANSI X12 files has been added. On the file open dialog, an option has been added to instruct Boxer to recognize an end-of-record character, after which a conventional line break will be added as the file is read. This function can also be activated (for the next file named) by using the -E command line option flag. This feature is useful for opening ANSI X12 files, and any other file format that uses a non-standard record ender. By default, the inserted line breaks will be written to the file when it is next saved. To override this behavior, use the File Properties dialog option to remove line enders when saving.

The File Tab context menu has a new **Open Containing Folder** option that can be used to open an Explorer window into the folder that contains the file being edited.

A set of commands has been added that allow **closed file tabs to be reopened more easily**. Right click on a file tab and choose "Undo Close Tab" to undo the most recent file closure. Options are also provided to undo all closed tabs, to selectively reopen closed tabs, or to clear the list of closed tabs. The last ten (10) closed file tabs are recorded. Clicking the middle mouse button in an empty area of the file tab zone is recognized as a shortcut gesture to undo the last closed file tab.

Boxer's file loading and saving has been enhanced so that it's now possible to **create a zero-length file**, or a file with just a few characters and no trailing newline. In the past, an empty line was assumed to always contain a line ender.

The Value at Cursor command now displays the **Unicode code point** for the character at the cursor, as well as the numeric or named HTML entity for that character.

- Visit our website for a comprehensive list of changes.

4 Command Reference (in menu order)

4.1 File Menu

4.1.1 New

Menu: File > New

Default Shortcut Key: Ctrl+N

Macro function: New()

The New command is used to create a new file for editing. The first new file opened in the edit session will be given the name `untitled.001`, and successive new files will use correspondingly higher file extensions to ensure uniqueness of the filename.

When a new file is first saved with either the Save or Save As command, a dialog box will appear so that a permanent name can be supplied for the file.

The size and position of the newly created window depends upon the nature of other open windows within Boxer. If other editing windows are maximized, the new window will also be created in maximized mode and will obscure the other windows below it. Otherwise, the new window will be created in 'normal' mode, and its size and position will be determined automatically by Windows.

Untitled files are not added to the Recent Files, and will not be reopened if an edit session is later restored.

You can request that new windows always be created in maximized mode with an option on the Configure | Preferences | Display options page. This option is titled *Auto-maximize new windows when created*.

If the desktop area within Boxer's main window is double-clicked, an empty/new file will be opened.

4.1.2 Picker

Menu: File > Picker

Default Shortcut Key: Alt+K

Macro function: FilePicker()

The File Picker command opens a dockable tool window alongside Boxer that can be used to open files for editing. The File Picker can remain open while Boxer is in use, making it easy to open new files for editing whenever the need arises. The treeview interface can display files from either the local PC or from PCs on attached networks. The File Picker also contains logical entries for "Desktop," "My Documents," and other conceptual locations on the local PC.

To open a file, double click on its name, or select it and press *Enter*. To open multiple files, select the files of interest and press *Enter*.

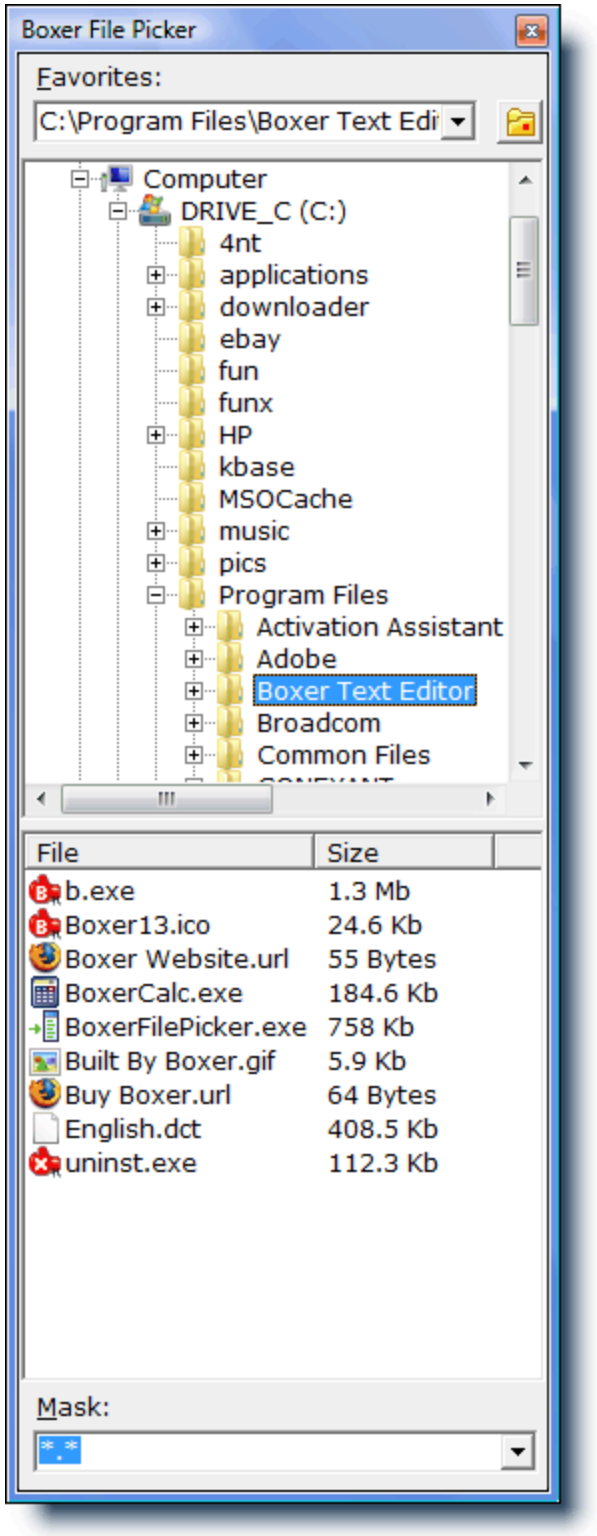
The right-click context menu contains options to open the selected file in its default application, or with an application of your choice. Commands to Cut, Copy, Rename and Delete files are also provided. Options are also provided to close the File Picker automatically when a file is opened, and/or when Boxer itself is closed.

A list of recent directories from which files have been opened is maintained in the *Favorites* list at the top of the File Picker window.

The *Mask* control permits the display of files to be filtered to show only a particular class of files.

The column headers in the file display area permit the file listing to be sorted by filename or file size.

By dragging the horizontal divider bar between the directory and file panes, you can control how much space is allocated to each pane.



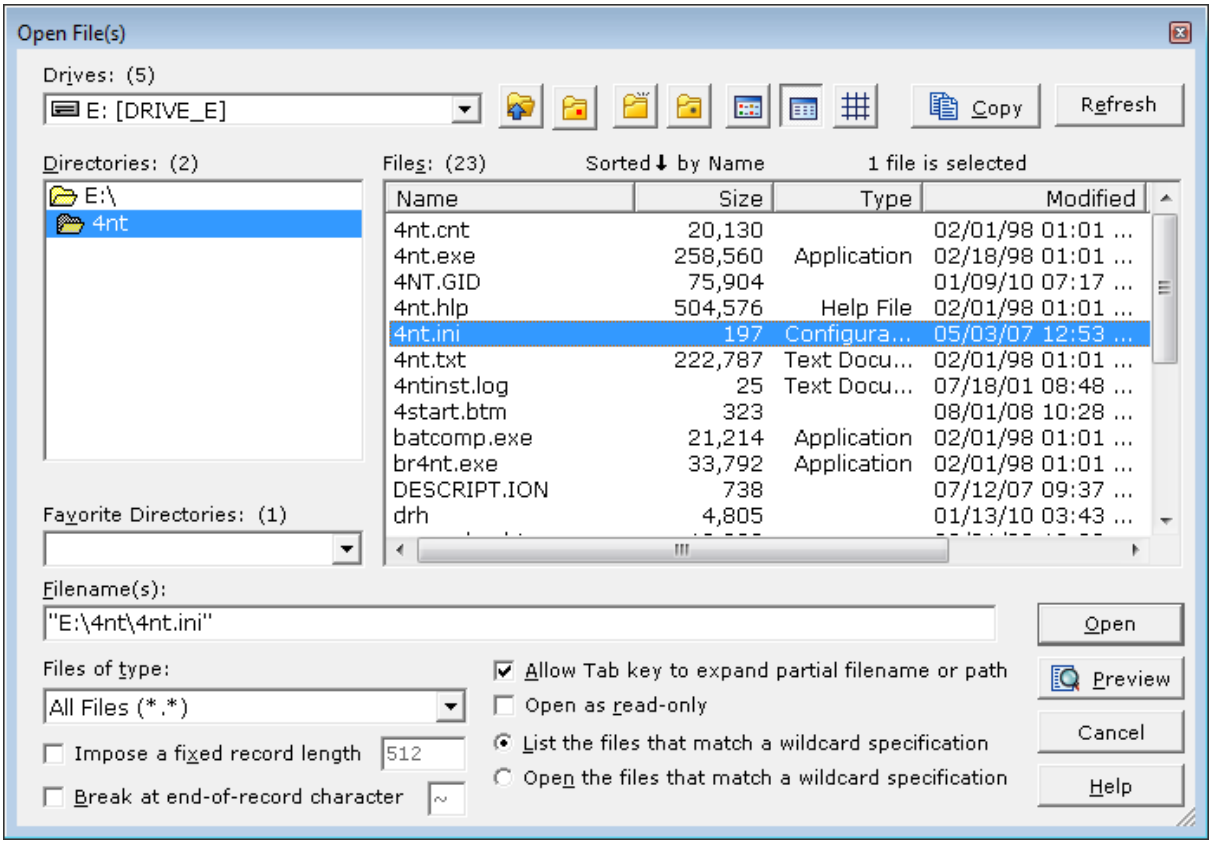
4.1.3 Open

Menu: File > Open

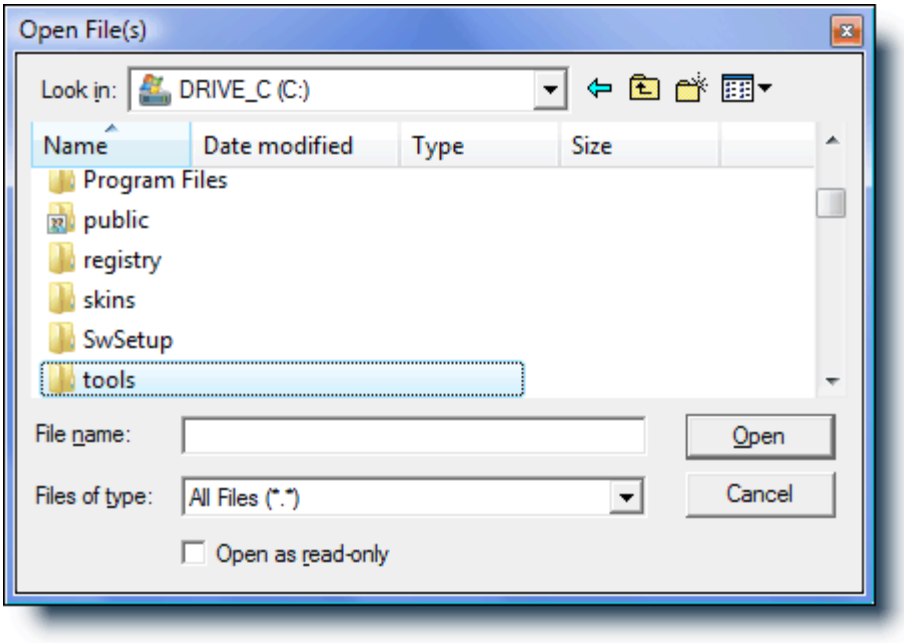
Default Shortcut Key: Ctrl+O

Macro function: Open()

The Open command is used to load one or more files for editing. By default, Boxer will present a custom File Open dialog with extra features not found in the standard Windows File Open dialog.



If you prefer to use the standard dialog, an option to do so can be found on the Configure | Preferences File I/O dialog page.



Standard Dialog or Custom Dialog

Opening a Single File
To select a single file for editing, click on the file's name within the display or type a filename in the *Filename* edit box. Click *Open* or press *Enter* to complete the selection.

Opening Multiple Files
To select multiple files for editing, depress the *Ctrl* key while clicking on a filename to add the new filename to the set of files to be opened. To select a series of adjacent files for editing, click on the first file in the series, and then depress the *Shift* key while clicking on the last file in the series.

Filtering Files
You can filter the files displayed by selecting a *file filter* from the drop-down list labeled *Files of type*. The file types which appear in this dialog are user-definable via the Configure | Preferences | File I/O options page. You can also filter the filenames displayed by typing a wildcard file specification (such as `m*.cpp` or `project.*`) into the *Filename* edit box.

Read-only Mode
To load a file for passive viewing, select the *Open as read-only* checkbox at the bottom of the dialog. Boxer will then prevent changes from being made to the file(s) during the edit session. This option can be used to protect against accidental changes to a file whose content needs to be viewed, but must not be altered.

Renaming Files
To rename a file, single-click once to select the item. Single-click again on the filename to place a text cursor into the filename. Type the new name for the file and press *Enter*
.

Deleting Files
To delete one or more files, first select the files to be deleted. Use Shift+Click to select a range of files, and/or Ctrl+Click to select discontiguous files. When the files are selected, press the *Delete* key to delete the files.

Important Note: the standard Windows dialog does **not** request confirmation before deleting a file, but it **does** place the deleted file in the Windows Recycle Bin. Boxer's custom dialog **does** require that file deletions be confirmed, but does **not** place the deleted file(s) in the Recycle Bin.

Window Sizes
The size and position of the newly created window depends upon the nature of other open windows within Boxer. If other editing windows are maximized, the new window will also be created in maximized mode and will obscure the other windows below it. Otherwise, the new window will be created in 'normal' mode, and its size and position will be determined automatically by Windows.

You can request that new windows always be created in maximized mode with an option on the Configure | Preferences | Display options page. This option is titled *Auto-maximize new windows when created*.

If you would prefer that changes made to the current directory not be reflected in future visits to the dialog, an option is available on the Configure | Preferences | File I/O options page. The option is titled *File I/O dialogs preserve current directory; ignore prior travel.*

Custom Dialog Features

The following features are available only with Boxer's custom File Open dialog

Resize Dialog
Boxer's File Open dialog is resizeable, and its size and position are remembered from session to session.

Current Directory
Use this icon to make the selected directory the current working directory.

Jump to the directory of the current file
Use this icon to make the directory of the current file the current working directory.

List View or Details View
Use these icons to select whether files are displayed in list view (simple), or with full details. Unlike the standard Windows dialog, the view mode is remembered from the previous dialog session.

Details View with Grid Lines
Click the icon to the left of the *Copy* button to enable a display mode in which light grid lines will by used to separate file data.

Copy Directory Listing
Use the *Copy* button to copy the entire directory listing to the current clipboard.

Refresh Display
Click the Refresh button to refresh the file display. This option could be used if another program or process has changed the file listing since the dialog was first opened.

Directory Count
A message at the top of the drive selection list shows the number of directories being displayed in that list.

File Count
A message at the top of the filename display shows the number of files in the current directory listing.

Selected File Count
A message at the top of the filename display shows the number of files currently selected.

File Sorting

File information can be sorted by any field that appears at the top of the table:

Click twice on a given field to reverse the direction of the sort. The current sort mode is displayed in a message at the top of the filename display. Unlike the standard Windows dialog, the sort mode and direction are remembered from the previous dialog session.

The *Name* heading also offers the ability to sort filenames by extension. Clicking the *Name* heading in succession yields the following sort modes: down by name, up by name, down by extension, up by extension.

You can change the width of any field by dragging the divider line between fields to a new position.

When sorting by Type, the file entries will be sorted first by type and then by file extension within the type.

Short Name Field
In addition to the standard *Name*, *Size*, *Type*, *Modified* and *Attribute* fields, Boxer's custom dialog also provides a *Short Name* field that displays the equivalent short filename for each file.

Favorite Directories
When files are opened for editing, the directory name is saved in the Favorites list. Use the Favorites drop-down list to quickly select a directory from among those recently used. Once a directory is selected its position in the list is elevated to position one.

Boxer will automatically add the directory of the current file to the Favorite Directories list when the Open dialog is activated.

When an existing entry in the Favorites Directory is found not to exist it will be automatically deleted from the list.

Filename Completion
Boxer's custom dialog supports filename completion. When typing a filepath or filename in the *Filename* field, press the *Tab* key to begin cycling through those directories or filenames that match your partial entry. When you see the directory name you were typing appear, continue typing the next few letters of the next directory name in the filepath. Press *Tab* to complete, as required. If you press accidentally press *Tab* too many times, press *Shift+Tab* to cycle backwards to a previously displayed name.

The function of the *Tab* key for Filename Completion can be disabled with the checkbox entitled *Allow Tab to expand partial filename or path*. In this case, the *Tab* key can again be used to move away from the *Filename* box.

List the files that match a wildcard specification
Use this option if you prefer that files matching a wildcard specification first be shown in the file list, and not opened directly. With this option, typing `*.XYZ` and clicking *Open* will cause the file list to be filtered to display only those files with a `.XYZ` extension.

Open the files that match a wildcard specification
Use this option if you prefer that files matching a wildcard specification be opened directly. With this option, typing `*.XYZ` and clicking *Open* will cause all files with a `.XYZ` extension to be opened for editing.

When entering a wildcard specification in the Filename box, the semi-colon (`;`) can be used to separate multiple file patterns. For example, to display (or open, depending on the above options) all files matching three different extensions, use the following syntax: `*.abc;*.def;*.ghi`

Impose a fixed record length as the file is read
This option can be used to impose a fixed-length record format on the file selected for opening. The record length is specified in the associated edit box. As the file is read, line breaks will be inserted at column 'value'.

This option is useful for viewing or editing files that contain fixed-length records. When the proper record size is entered, Boxer will display the file in a meaningful format, with one record per line.

Note that if the file is saved, line enders will be added. The File | Properties dialog has an option to request that line enders be removed when the file is saved.

The -F command line option flag can also be used to impose a fixed record length on the file being opened.

Because this option is dangerous to use on files that do not contain fixed-length records, the checkbox state and record length values are not recalled from the previous dialog session. Rather, the checkbox always reverts to the safe, unchecked

state when the dialog is opened.

Break at end-of-record character

his This option provides support for editing files that use an end-of-record character, such as ANSI X12 files. When this option is active, Boxer will watch for the designated end-of-record character as the file is read, after which a conventional line ender will be added. By default, the inserted line enders *will* be written to the file when it is saved. To override this behavior, use the File Properties dialog to request that line enders be removed at File-Save time.


The -E command line option flag can also be used to activate this option for the next-named file on the command line.

File Preview

The *Preview* button enables you to view the content of a file without actually opening the file for editing. A pop-up window will appear that displays the opening content of the file. From this view you can decide if the file in question is the one you would like to open. The pop-up Preview window disappears automatically as soon as the mouse cursor is moved significantly.


Binary files are not eligible for display with the Preview function.

Context Menu

Right-clicking on a selected file (or files) will present the File Open dialog's context menu. The following commands are provided: Open, Preview, Select All, Copy, Rename and Delete. The Rename and Delete commands provide an alternative method to perform these functions, which are described in the section above. The Copy command copies the selected file(s) to the Windows clipboard in a manner that allows them to be pasted into Explorer, effecting a file copy. Other applications may also be able to recognize the 'Explorer Copy' clipboard format and behave accordingly.

Notes


The directory list control used by the custom dialog (known to programmers as TComboBox) does not show abstract entries such as the Desktop, Network Neighborhood, Shared Drives, etc. TComboBox is limited to the display of physical drives. All of the abstract entries have a corresponding location on the physical disk drive, if you know where to look. If you would prefer to see the abstract entries, you can revert to the standard File Open dialog using an option on the Configure | Preferences File I/O dialog page.

4.1.4 Open Hex

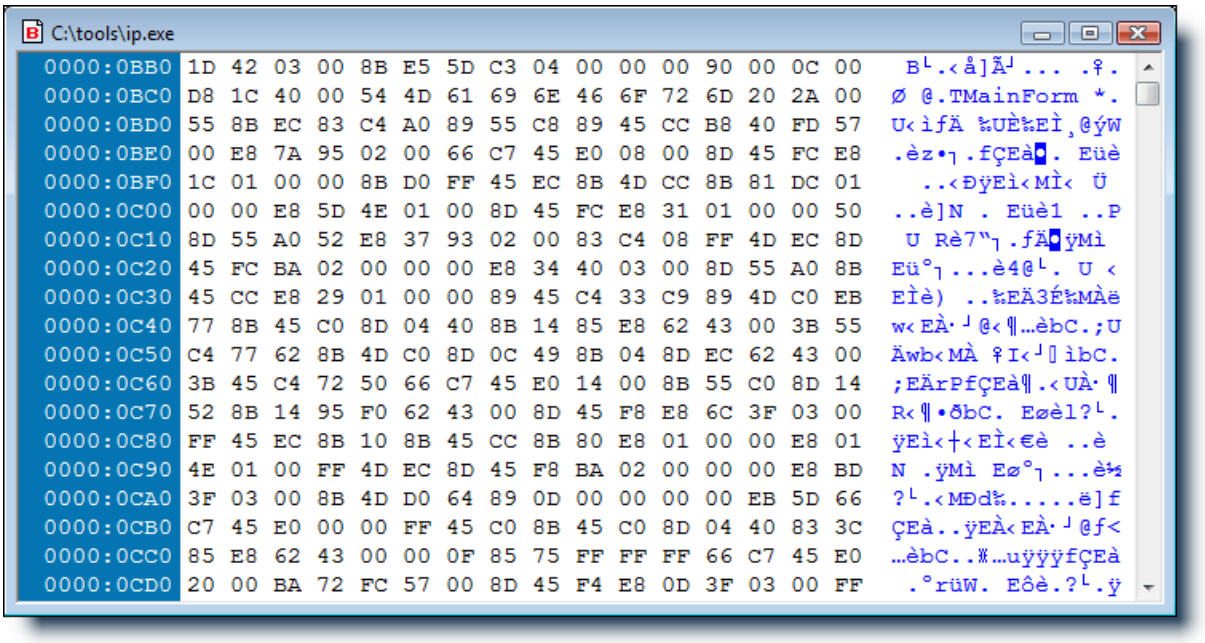
Menu: File > Open Hex

Default Shortcut Key: Ctrl+Alt+O

Macro function: OpenHex()

The Open Hex command is used to open a file in hex mode for viewing or editing. After selecting the file to be opened from the Open dialog, a new window is created and the

file is displayed within the window:



Files opened with Open Hex are displayed in a special format which has three sections. At the left, in the line number zone, the byte offset into the file is shown in *hexadecimal* format. In the center, sixteen data bytes are displayed as two-byte hexadecimal values. At the far right the same sixteen bytes are displayed as ASCII characters, except in cases where the character in question cannot be so represented.

☞ The representation of the sixteen characters at the right depends upon whether an ANSI or OEM screen font is in use. The screen font can be changed with the Screen Font command.

To edit a value, position the text cursor over the two-digit hex value that is to be changed and type a new hex value. Alternatively, the cursor can be placed in the ASCII area at the right and characters can be keyed directly from the keyboard. The Tab key can be used to jump between equivalent positions in the hex and ASCII display areas.

In most cases, *binary files* should not be shortened or lengthened. Doing so will often invalidate the format of the file. If you are editing a file whose length *can* be safely altered, and you wish to do so, the right-click hex mode context menu provides options for inserting and deleting one or more bytes into/from the file.

When editing in hex mode, the Find and Replace commands will each present a special dialog which is designed for use in hex mode. In hex mode, these commands will permit strings to be entered as either a sequence two-digit hex codes, or as conventional text strings.

☞ Boxer will automatically use the Open Hex command when asked to open files with the extension `.COM`, `.EXE` or `.DLL`. Files with these extensions are assumed to be *binary files*, and cannot be edited by Boxer in conventional text mode.

The View | Hex Ruler command displays a hex ruler across the top of the screen.

The Go to Byte Offset command is sensitive to Hex Mode, and can be used to jump quickly to a specified offset within the file.

See the Open command for details on using Boxer's custom file open dialog and the standard Windows file open dialog.


The maximum size for a file opened in hex mode is smaller than the usual file size limit. The limit is approximately 478 MB. This limit derives from the fact that the display of 16 characters in hex mode requires approximately 80 characters of storage space.

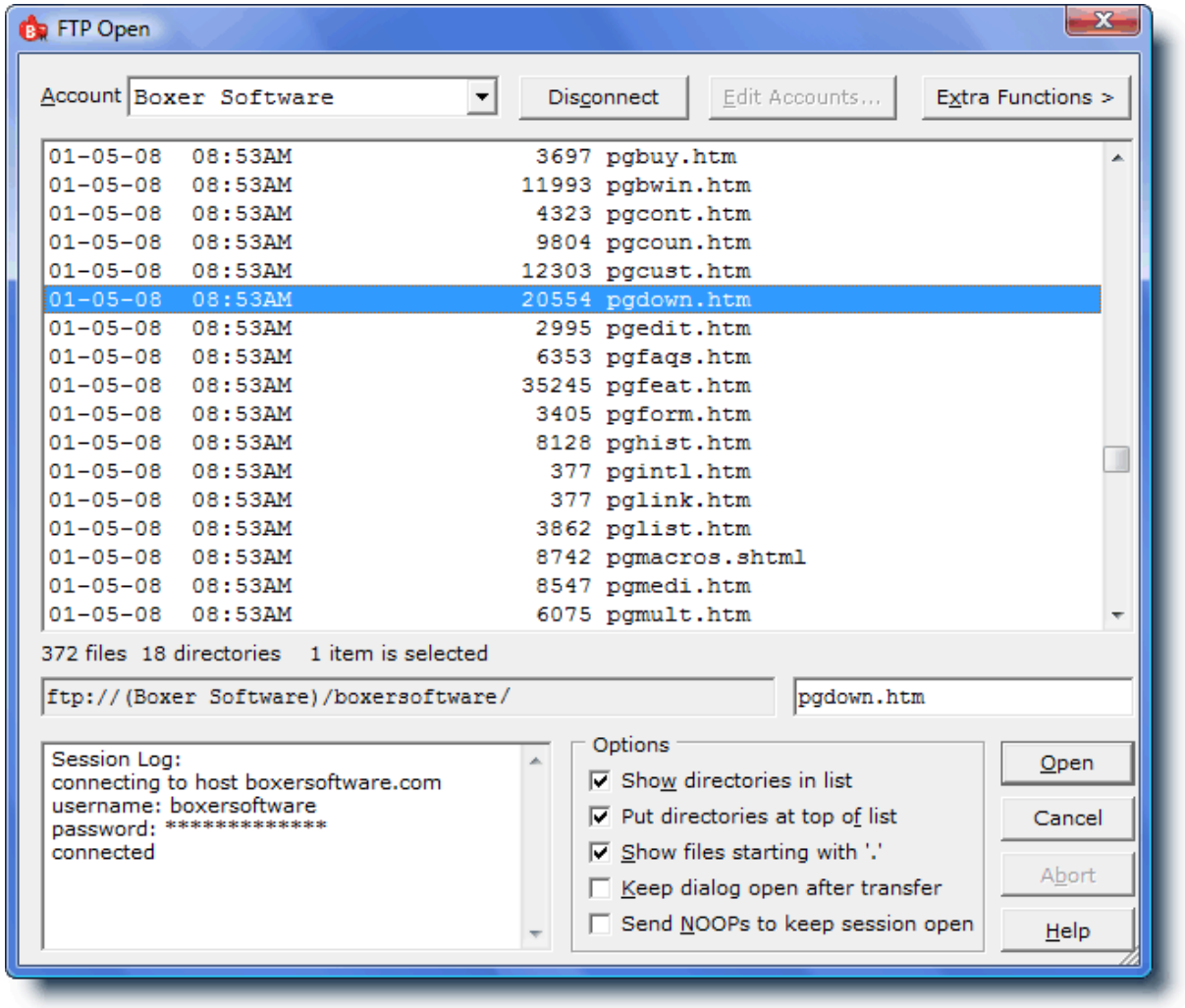
4.1.5 FTP Open

Menu: File > FTP Open

Default Shortcut Key: Shift+Alt+O

Macro function: none

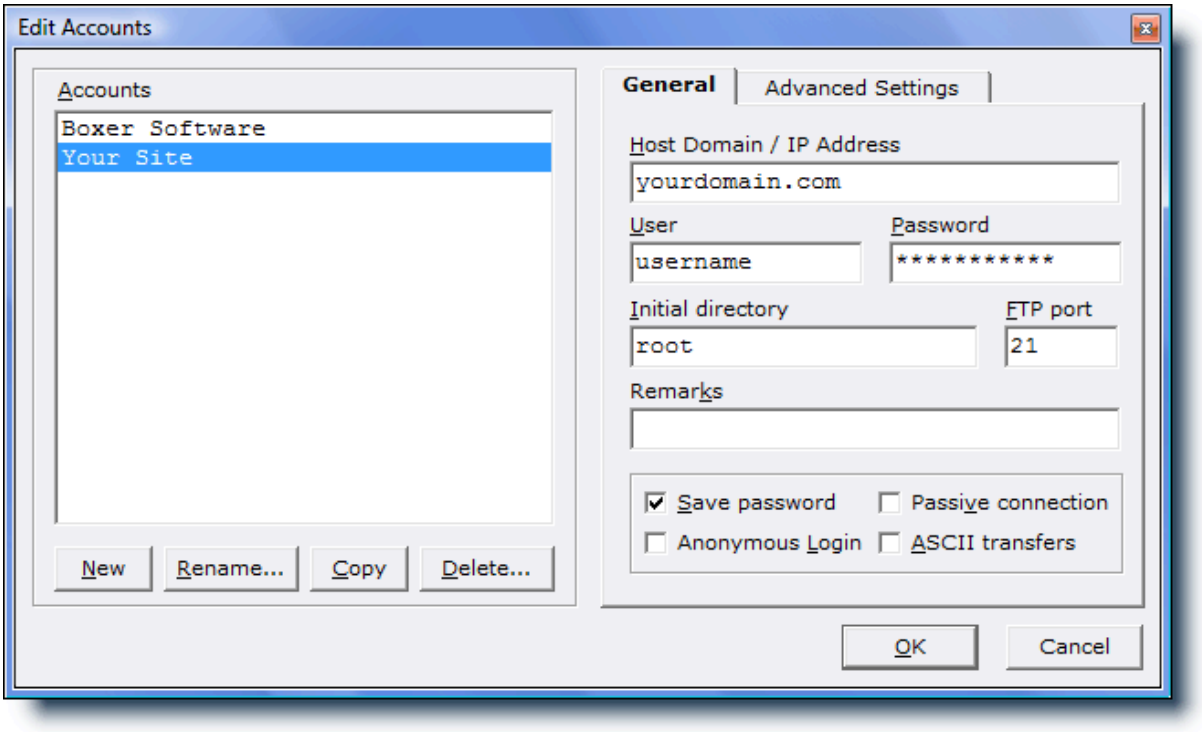
The FTP Open command provides the ability to open files on a remote computer system for viewing and/or editing. A custom dialog is used to display the files on the remote system and to provide access to the various FTP functions:



This help topic is not intended to be a primer on the subject of FTP (File Transfer Protocol) communications. Some familiarity with FTP is assumed on the part of the reader.

Edit Accounts Dialog

The first step to opening files on a remote system is to provide Boxer with the information necessary to establish an FTP connection. The *Edit Accounts* button will present the following dialog:



Initially the dialog will not show any accounts in the *Accounts* listbox at the left. Click *New* to create a new account. Use the *Rename* button to rename the new account as desired.

The *Copy* button can be used to make a copy of the selected account. The *Delete* button can be used to delete a selected account.

Host Domain / IP Address
This field holds the domain name or IP address of the site you wish to connect to.

User
This field holds the user name for the account you have at the remote site. Up to 40 characters are permitted.

Password
This field holds the password for the account you have at the remote site. For security purposes, the field will be displayed with asterisks as it is typed. If the password field is left blank, you will be prompted for the password at connect-time. Up to 40 characters are permitted

☞ If the *Save password* option is selected, the password you enter will be stored in the *Windows registry* along with Boxer's other settings. For security purposes, the password entry will be encrypted in the registry.

Initial Directory
This field holds the name of the directory you would like to be positioned in on the remote system.

If the system you connect to has several directories with different functions, you might choose to create several accounts that differ only in the *Initial Directory* field. This makes it easier to connect directly to the required directory location. The *Copy* button makes it easy to create copies of existing accounts.

FTP port
This field holds the port number to be used for the FTP connection. Unless you have reason to do otherwise, the value should remain at its default setting of 21.

Remark
This field can be used to add a comment about the FTP account being configured.

If you prefer that your password be visible, and security is not an issue, the Remark field can be used to note the password.

Save Password
Use this option to dictate whether or not the *Password* field will be saved from session to session. If this option is *not* selected, the value of the *Password* field will persist only for the current session.

Anonymous Login
Use this option to configure for a system which permits Anonymous login. Selecting this option causes the *User* field to be set to 'anonymous'. The *Password* field is not required, but some systems suggest that your email address be supplied as the password.

Passive Connection
Use this option to request that the FTP session use a passive connection. A passive connection may be required by some firewalls.

ASCII transfers
Use this option to indicate that file transfers are to be performed in ASCII mode. If this option is not selected transfers will be made in Binary mode.

Line ender conversion issues can arise when transferring files from Unix systems. If the results you are seeing are unsatisfactory, try switching the sense of the *ASCII transfers* option.

Advanced Settings
The advanced settings dialog tab is reserved for future use.

FTP Open Dialog

Account
The *Account* list provides a drop-down list of all defined accounts. Select the desired account before clicking *Connect*.

Connect / Disconnect
The *Connect* button is used to initiate an FTP connection. Once the connection is established, the label of this button changes to *Disconnect*.

File List
The *File List* display shows the names of the files on the remote system. The format of this list will depend on the remote system. Boxer does not coerce the data supplied by the remote system into a new format. Likewise, the method used to indicate directory names will vary from system to system. The most common formats use either a 'd' in column one, or the string '<DIR>' somewhere near the filename.


File entries on the remote system which are 'links' or 'redirects' will not be shown in the *File List*. The knowledge of where a link points to is maintained on the remote system. Opening a link via FTP does not have the expected results.

Activity Log
At the lower left of the dialog is the *Activity Log*. This scrolling list maintains a record of the functions that have been performed during the FTP session. The log can be cleared with the *Clear Log* button that appears in the *Extra Functions* panel.

Show directories in list
Use this option to dictate whether or not directory names should appear in the *File List*.

Put directories at top of list
Use this option to dictate whether or not directory names should be placed at the top of the *File List*. If directories are not placed at the top they will appear sorted among the filename entries.

Show files starting with '.'
Use this option to control whether or not filenames that begin with a period should be displayed in the *File List*. On some systems such files are used for configuration and should not be disturbed.

Keep dialog open after transfer
Use this option to control whether the FTP dialog should remain open after the transfer is completed. This may be desirable if multiple files are to be opened from different remote directories.

Send NOOPs to keep session open
Use this option to request that an FTP session be kept open by sending NOOPs (NO Operation commands) to the remote system. This option could be used to defeat a system which enforced an automatic logout after a period of inactivity.


Some systems will not interpret NOOPs as legitimate FTP activity and will proceed with automatic logout.

Extra Functions
The Extra Functions button will toggle on and off a panel that displays additional FTP functions. The following functions are available:

Change To
Use this function to change to the selected directory.


The *Enter* and *Right Arrow* keys can also be used to descend into a selected directory.

Up Dir
Use this function to change to the parent directory.

 The *Left Arrow* can also be used to ascend to a parent directory.

Make Dir
Use this function to create a new directory.

Rename
Use this function to rename a remote file or directory.

Delete
Use this function to delete a remote file or directory. A confirmation will be required before the deletion is performed.

Power Copy
The *Power Copy* function will copy all local files that are open in the editor to the current remote directory. If any of these files have been modified they will be saved automatically. A confirmation is required before the *Power Copy* operation is performed.

 This function can be very useful when editing local copies of website files.

Refresh View
Use this function to refresh the *File List* display. This might be required if you suspect that another program or process has made changes to the remote system that affect the file listing.

Clear Log
Use this function to clear the content of the *Activity Log* window.

Notes

Once a remote file is opened for editing, the name and path of that file will be displayed in the title bar of its edit window. The FTP Filepath includes the information that Boxer needs in order to be able to save the file, or to reopen it at a later time. An FTP Filepath has the form:

```
ftp://(AccountName)/remote_dir/remote_filename.ext
```

Many different areas of Boxer have been enhanced to recognize and process FTP filepaths:

File Save
When the Save command is issued an FTP connection to the remote system will be established automatically so that the file can be saved.

Command Line
When an FTP Filepath appears on Boxer's command line it will automatically be opened for editing when the edit session begins.

Wildcard file specifications are not supported in this context.

Project Files
An FTP Filepath can be placed within a Project file so that it can be opened along with other files named in the Project file.

Most Recently Used Files
When an FTP Filepath is recalled from the Recent Files list near the bottom of the File menu, it will be opened automatically.

Restored Session
When a previous edit session is restored, any remote files that were open in that session will be opened automatically.

Filename at Cursor
If an FTP Filepath is found beneath the text cursor, the Open Filename at Cursor command will open the file automatically.

Spaces are permitted within an FTP account name, but you might wish to avoid using them. Doing so will make it easier to use FTP Filepaths on the command line, or with the Open Filename at Cursor command, since double quoting of the filepath will not be required.

4.1.6 Open Other -> Header File

Menu: File > Open Other > Header File

Default Shortcut Key: Ctrl+H

Macro function: OpenHeaderFile()

The Open Header File command provides a method to quickly open the header file which is associated with the file being edited. The most common use of this command will be for programming, but the command's utility could be extended to any file extension pairs which are related in the same way.

Example: while editing in the C++ file `main.cpp`, issuing the Open Header File command will cause `main.hpp` to be loaded. Likewise, if `main.hpp` is being edited, issuing the Open Header File command will cause `main.cpp` to be loaded. If the associated file is already loaded within the editor, it simply becomes the active window. Issuing the command repeatedly will allow you to toggle back and forth between the two associated files.

Boxer comes with several header file associations pre-defined for common programming languages. Additional header file extension pairs can be defined on the Configure | Preferences | File I/O options page. The option is titled *Open Header File extensions*.

4.1.7 Open Other -> Filename at Cursor

Menu: File > Open Other > Filename at Cursor

Default Shortcut Key: Ctrl+L

Macro function: OpenFilenameAtCursor()

Open Filename at Cursor is a timesaving command which allows the filename beneath the text cursor to be loaded for editing. Simply issue this command while the cursor is atop a filename and the file will be loaded into a new editing window. If the file does not exist, a new file with that name will be created in the current directory.

If selected text of a suitable size is present, the selected text will be used as the filename to be opened. If the filename to be opened contains spaces, it *must* be selected to ensure the full filename, including embedded spaces, will be used.

This command will be disabled when the text cursor is sitting upon a text string which could not be a valid filename, such as a series of spaces.


If the filename at the cursor does not exist in the current working directory, but does exist in the directory of the currently edited file, it will be opened from that directory instead.

4.1.8 Open Other -> System Files

Menu: File > Open Other > System Files

Default Shortcut Key: none

Macro function: OpenSystemFiles()

The Open System Files command automatically loads a variety of operating system files into the editor. On Windows 95, Windows 98 and Windows Me, the following files are opened:

```
autoexec.bat
config.sys
system.ini
win.ini
```

On Windows NT, 2000 and XP, the following files are also opened:

```
config.nt
autoexec.nt
```

If the command is invoked while one or more system files are already open, an option is provided to close these files.


It is advisable to exercise extreme caution when editing files of this nature, and to

always keep a backup copy of the original file.

4.1.9 Open Other -> File in Browser

Menu: File > Open Other > File in Browser

Default Shortcut Key: Ctrl+B

Macro function: OpenFileInBrowser()

The Open File in Browser command will attempt to load and display the current file within your Internet browser program. This command is useful for reviewing the effect of changes made to HTML files, or any other files which are eligible for viewing within an Internet browser..

For those comfortable with HTML coding, this command permits Boxer to be used as a powerful HTML editor with true *WYSIWYG* display. The procedure to use is as follows: make your HTML changes within Boxer, save the file, press *Ctrl+B* to activate the browser window, and then click 'Reload' or 'Refresh' to load the latest changes from disk. Once the browser has been opened you can continue to use *Ctrl+B* from within Boxer to switch back to the browser window. Some users may find this method preferable to using a dedicated HTML editor, since many of these editors lack a true WYSIWYG display and/or comprehensive editing features.

Boxer decides whether a file is eligible for display within a browser by checking its list of eligible file extensions. This list can be edited from the Configure | Preferences | File I/O options page. The option is titled *Open File in Browser extensions*.


In order to launch your browser, Boxer relies upon the file associations which exist within the operating system between the browser and its eligible file types. Most browsers establish these file associations during their setup procedure. If you find that certain file extensions do not result in your browser being launched, it is due to the absence of the required file association(s) within the operating system, and not due to any shortcoming in Boxer itself.

4.1.10 Open Other -> Email at Cursor

Menu: File > Open Other > Email at Cursor

Default Shortcut Key: Ctrl+E

Macro function: OpenEmailAtCursor()

The Open Email at Cursor command will attempt to launch your email program to send a message to the email address beneath the text cursor. Your email program will display the email address in its '*to*' field and prompt for a subject. You can then compose your message and send it in the usual way.

This command can also be invoked by double clicking with the mouse on an email address which appears within text. The mouse cursor will change to the pointing hand when atop an email address to indicate that the address has been recognized.

In order to launch your email program, Boxer relies upon the operating system shell's ability to process a 'mailto' directive. When an email client program is installed, it typically establishes itself as the program which is called by the shell to process the mailto command. If you find that your active email program is not launched by Boxer, or if some other inactive email program is launched instead, it's probably because your active email program did not establish itself to be the program that processes mailto commands. This situation cannot be remedied by Boxer, and is not due to any shortcoming in Boxer. You might consult the documentation of your email program, or contact its vendor.

4.1.11 Open Other -> URL at Cursor

Menu: File > Open Other > URL at Cursor

Default Shortcut Key: Ctrl+U

Macro function: OpenURLAtCursor()

The Open URL at Cursor command will attempt to launch your Internet browser to view the *URL* address beneath the text cursor.

This command can also be invoked by double clicking with the mouse on a URL which appears within text. The mouse cursor will change to the pointing hand when atop a URL to indicate that the address has been recognized.

In order to launch your Internet browser, Boxer relies upon the operating system shell's ability to open an Internet address. When an Internet browser is installed, it typically establishes itself as the program which is called by the shell to open such addresses. This is true of all common browsers you are likely to encounter. If you find that your Internet browser is *not* launched by Boxer, or if some other inactive browser is launched instead, it's because your active browser has not established itself as the one that processes the 'open' request from the operating system shell for Internet addresses. This situation should be rare, cannot be remedied by Boxer, and is not due to any shortcomings in Boxer.

Boxer also supports opening local files with URLs of the form:

```
file://c:\website\index.html
```

4.1.12 Open Other -> Program at Cursor

Menu: File > Open Other > Program at Cursor

Default Shortcut Key: none

Macro function: OpenProgramAtCursor()

The Open Program at Cursor command can be used to open the program or document that appears at the text cursor. For example, if the filepath to a `.PDF` file appears at the text cursor, this command will open that file in Acrobat Reader. If a `.DOC` file appears at the cursor, the file will be opened in Microsoft Word.

This command relies on the underlying ability of the operating system to identify and locate the program that is associated with a given file type. If a document does not have an associated program, it cannot be opened. See File Associations for more details.

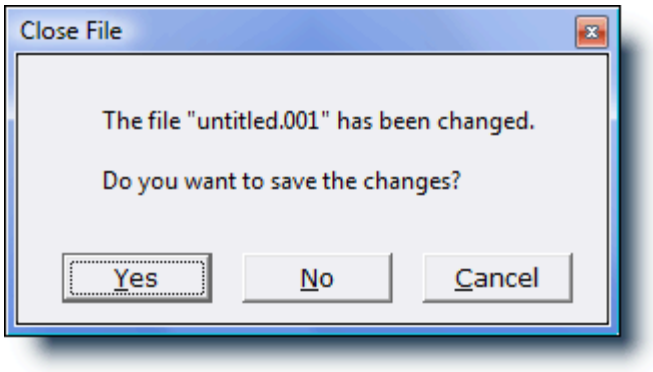
4.1.13 Close

Menu: File > Close

Default Shortcut Key: Alt+X

Macro function: Close()

The Close command is used to close the file in the active editor window. If unsaved changes have been made to the file, a dialog box will first appear to alert you to this fact. You will then be able to choose whether to save the changes before closing, close without saving, or cancel the Close operation altogether.



You can quickly tell whether a file has unsaved changes by looking for an asterisk (`*`) to the left of its name in the title bar, or on its File Tab.

A file can also be closed by clicking the 'X' box in the upper right corner of its window.

When a file's window is maximized, the 'X' box will be located at the far right of the main menu bar.

If you would prefer that Boxer be minimized automatically when the last file is closed, there is a checkbox on the Configure | Preferences | Other options page to achieve this. The option is titled *Minimize Boxer when closing last file.*

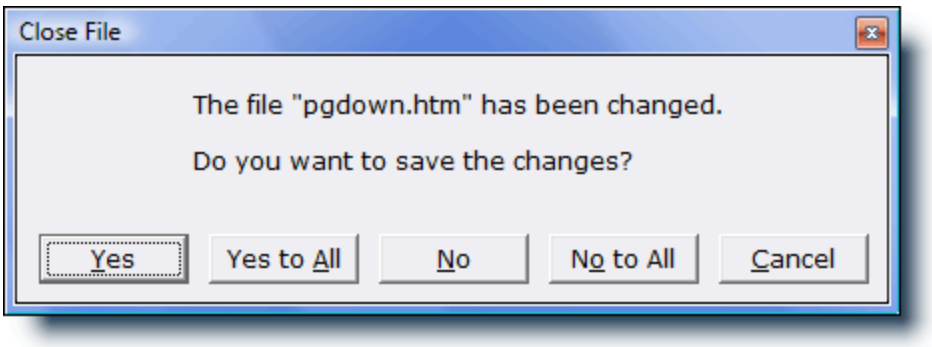
4.1.14 Close All

Menu: File > Close All

Default Shortcut Key: none

Macro function: CloseAll()

The Close All command is used to close all open files within the editor. If unsaved changes have been made to any file, a dialog box will appear for each such file to alert you to this fact. You will then be able to choose whether to save the changes before closing, close without saving, or to cancel the Close All operation. If Close All is issued when more than one file is modified, *Yes-to-All* and *No-to-All* buttons are provided to save time:



You can tell quickly whether a file has unsaved changes by looking for an asterisk (*) to the left of its name in the title bar, or on its File Tab.

If you would prefer that Boxer be minimized automatically when the last file is closed, there is a checkbox on the Configure | Preferences | Other options page to achieve this. The option is titled *Minimize Boxer when closing last file*.

The File | Close All command is functionally equivalent to the Window | Close All command, since each file resides in its own window.

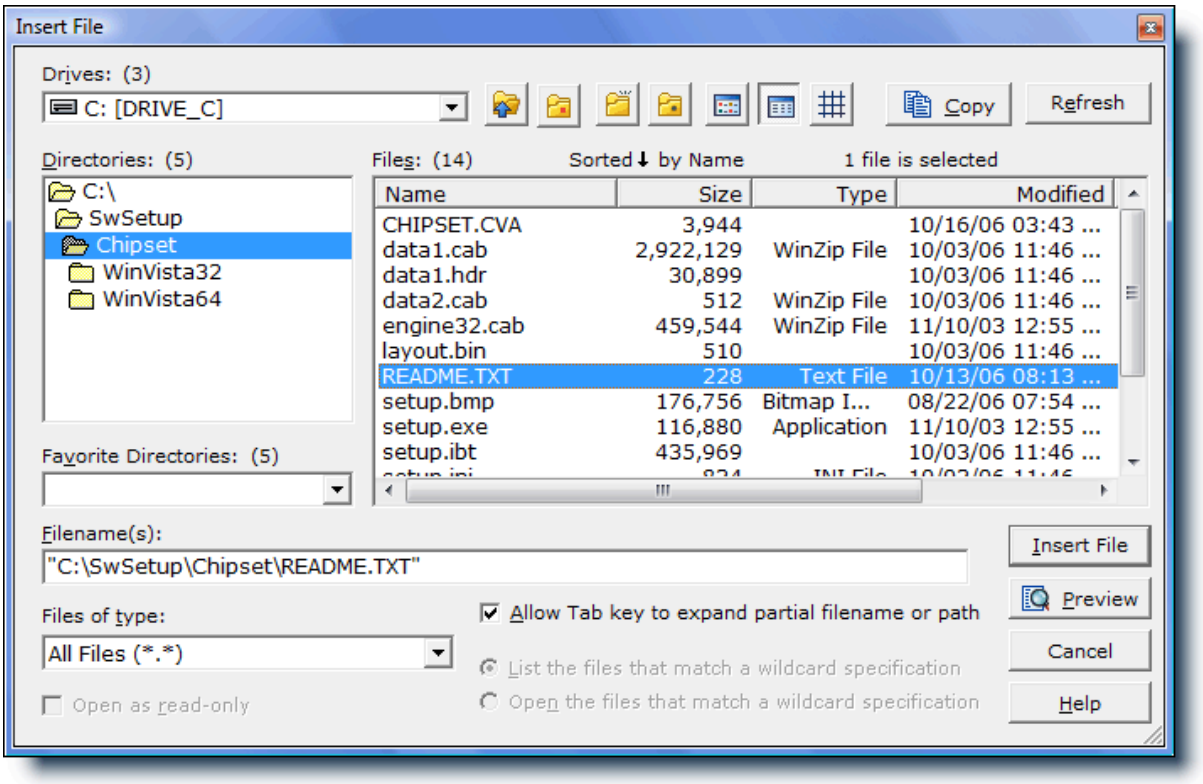
4.1.15 Insert

Menu: File > Insert

Default Shortcut Key: Ctrl+I

Macro function: InsertFile()

The File Insert command is used to insert (some may say 'import') the content of another file at the current location of the text cursor. The file open dialog is presented for selecting a file and, after its selection, the content of the file will be placed at the cursor.



Any text file can be selected for insertion (subject to Sizes and Limits), even one which is being edited in another editor window.

See the Open command for full details on using both the custom and standard Windows File Open dialogs.

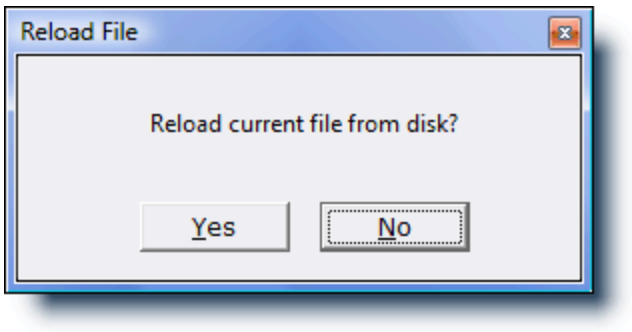
4.1.16 Reload

Menu: File > Reload

Default Shortcut Key: Shift+Ctrl+O

Macro function: ReloadFile()

The Reload command can be used to reload the current file from disk, thereby losing any unsaved changes in the file. Because of the potential for losing changes accidentally, a dialog box will appear to confirm your intention to reload.



The most common use for this command is to restart the editing of a file after having made a large number of unwanted changes. The Undo command can be used to step back through the changes made, but its limits can be exhausted if a large number of changes are made. In that case, the Reload command must be used.

4.1.17 Save

Menu: File > Save

Default Shortcut Key: Ctrl+S

Macro function: Save()

The Save command is used to write the contents of the current file to disk. If the file being edited is a new file with a temporary untitled name, the Save As command will be performed automatically so that a name can be provided. The Save command will be disabled when there are no changes in the current file that need to be saved.

☞ If the Save command is issued while text is selected, a dialog box can appear to get the name of the file to which the selected text should be saved. This option is off by default, but can be enabled on the Configure | Preferences | File I/O options page. The option is titled *File Save performs Save Selection As, when text is selected*. This option page also contains other configuration options which relate to saving files.

4.1.18 Save All

Menu: File > Save All

Default Shortcut Key: Shift+Ctrl+S

Macro function: SaveAll()

The Save All command can be used to write the contents of all files with unsaved changes to disk. If any of the files being edited are new files with a temporary untitled name, the Save As command will be performed automatically so that a name can be provided. The Save All command will be disabled when there are no changes in any open files that need to be saved.

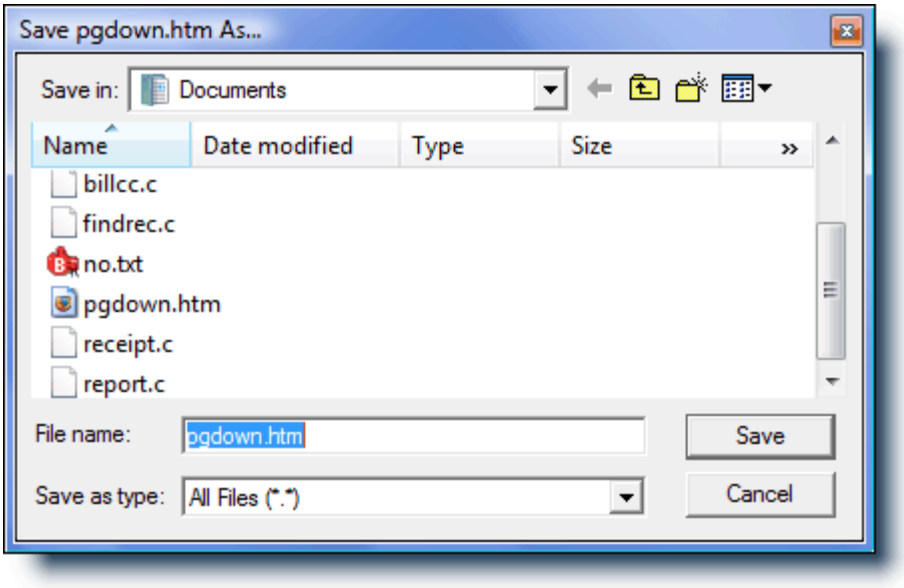
4.1.19 Save As

Menu: File > Save As

Default Shortcut Key: F12

Macro function: SaveAs()

The Save As command is used to save the current file to disk using a new filename. A standard Windows File Save dialog box will appear so that a new name can be specified. The file will remain on disk under its old name (except when an untitled file is being saved), and a copy of the file will be saved to the new name provided. Boxer will then record the file's new name so that all future save operations are made to the new name.



If you would like to change either the line ender style (PC, Macintosh, or Unix), or the file encoding (ASCII, UTF-8, UTF-16), visit the File Properties dialog before using the Save As command to save the file.

Boxer will not automatically add a file extension to the filename you provide; you should add the desired file extension yourself.

If the Save As command is used to save a file which is being viewed in read-only Hex Mode, the hex view of the file--and not the file's true content--is what will be saved to disk. If a file was opened for editing in hex mode, then Save As will create a copy of the file's actual content.

4.1.20 FTP Save As

Menu: File > FTP Save As

Default Shortcut Key: Shift+Alt+F12

Macro function: none

The FTP Save As command is used to save the current file to a remote computer using a new filename. The file will remain on disk under its old name (except when an untitled file is being saved), and a copy of the file will be saved to the new name and location provided. Boxer will then record the file's new name so that all future save operations are made to the new name.

Boxer's FTP dialog will be used to initiate the connection to the remote computer. For full details about this dialog please see the FTP Open command.


If you are editing a remote file and wish to save a copy locally, use the Save As command.

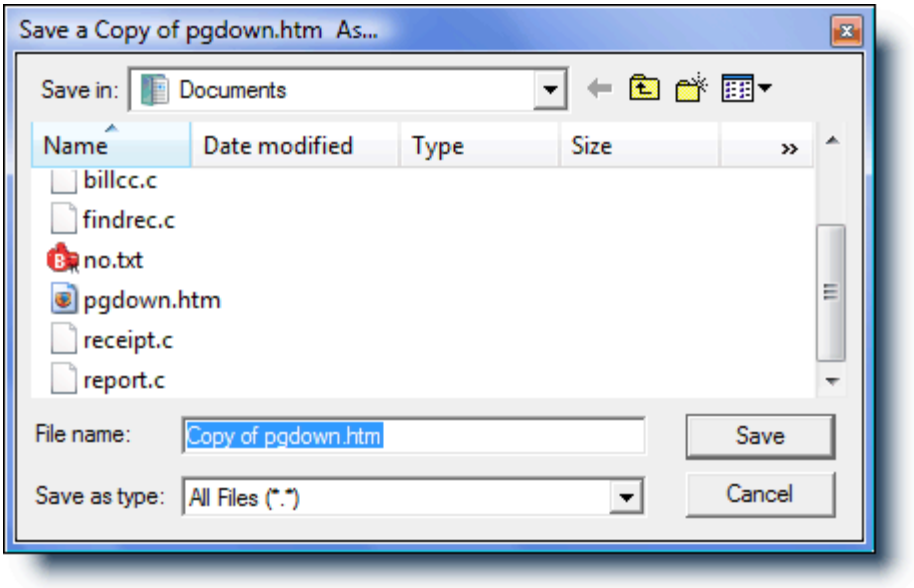
4.1.21 Save a Copy As

Menu: File > Save a Copy As

Default Shortcut Key: none

Macro function: SaveACopyAs()

The Save a Copy As command is used to save the current file to disk under a new filename. A standard File Save dialog box will appear so that a new name can be specified. Unlike the Save As command, the editor does not record the new filename for use by future save operations. This command can be useful for creating progressive copies of a file during a long edit session so that a change history is created, or at anytime when a copy might be useful.



Boxer will not automatically add a file extension to the filename you provide; you should add the desired file extension yourself.

If the Save a Copy As command is used to save a file which is being viewed in read-only Hex Mode, the hex view of the file--and not the file's true content--is what will be saved to disk. If a file was opened for editing in hex mode, then Save a Copy As will create a copy of the file's actual content.

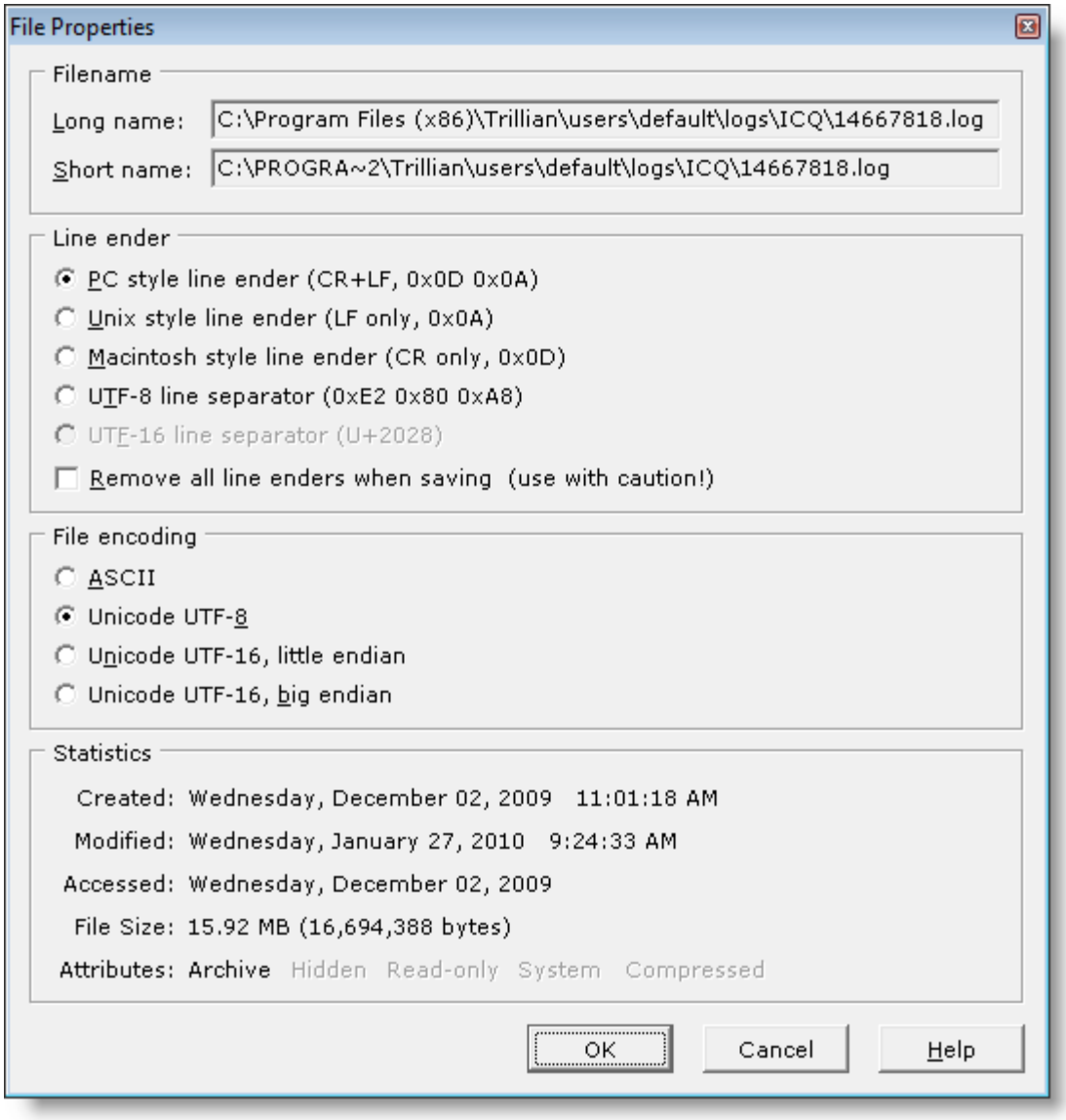
4.1.22 File Properties

Menu: File > Properties

Default Shortcut Key: none

Macro function: FileProperties()

The File Properties command displays a dialog box containing information about the file on disk which is associated with the editor's current file. The file's long and short names, create/modify/access times, file size and file attributes are all displayed. Because this command reports information about the current file's disk file, it will be disabled when editing a file which has yet to be written to disk.



Line ender

PC
Use this option to save files with a PC style line ender (CR+LF).

Unix
Use this option to save files with a Unix style line ender (LF only).

Macintosh
Use this option to save files with a Macintosh style line ender (CR only).

UTF-8 line separator
Use this option to save files with a UTF-8 line separator. This option is only valid when file encoding is set to Unicode UTF-8. Note that this line ender sequence uses three bytes: hex 0xE2, 0x80 and 0xA8.

UTF-16 line separator
Use this option to save files with a UTF-16 line separator (U+2028). This option is only valid when file encoding is set to one of the Unicode UTF-16 encoding options.

Boxer can read files with any type of line ender, and it will make note of the line ender as the file is read. The above options can be used to force a file to be written with a specific line ender.

An option to set the line ender styles that is used for newly created files appears on the Configure | Preferences | Editing 1 dialog page.

Remove all line enders when saving
This option can be used to ensure that no line enders are written to the output file when it is saved. This option should **not** be used for conventional text files, as all line-related formatting will be lost. Rather, this option can be useful when saving a file that contains fixed length records, if its file format requires that line enders not appear within the data stream. To load such a file for viewing or editing, Boxer's File Open dialog contains an option to impose a fixed length record format onto the file being opened. The -F command line option is also available for this purpose.

File encoding

ASCII
Use this option to save the current file with ASCII encoding. This is the standard file format for most text files. No header characters or null characters will appear in the output file.

UTF-8
Use this option to save the current file with UTF-8 encoding. This is a Unicode format that uses between one and four bytes to encode each character unambiguously. Null characters do not appear in UTF-8 encoded files.

Though it is not displayed on-screen in the editor, the three-byte UTF-8 Byte Order Mark (0xEF 0xBB 0xBF) will be applied when a UTF-8 encoded file is saved to disk. These bytes will appear as the first three bytes in the file.

UTF-16, little endian
Use this option to save the current file with UTF-16 little endian encoding. This is a Unicode format that uses two bytes to encode each character that appears in the file. The 'little endian' designator refers to the byte order in the output file. The little endian version of UTF-16 is popular on Windows-based PCs. Null characters will almost certainly appear in the output file.

Though it is not displayed on-screen in the editor, the UTF-16 Byte Order Mark (U+FEFF) will be applied when a UTF-16 encoded file is saved to disk. In a UTF-16 little endian file, the first two bytes will be 0xFF 0xFE.

UTF-16, big endian
Use this option to save the current file with UTF-16 big endian encoding. This is a Unicode format that uses two characters to encode each character that appears in the file. The 'big endian' designator refers to the byte order in the output file. Null

characters will almost certainly appear in the output file.

Though it is not displayed on-screen in the editor, the UTF-16 Byte Order Mark (U+FEFF) will be applied when a UTF-16 encoded file is saved to disk. In a UTF-16 big endian file, the first two bytes will be 0xFE 0xFF.

See the help topic Unicode Files for full information about Boxer's handling of Unicode files.

The active *code page* can be viewed using the System Info option on Boxer's About dialog.

An option to set the default line ender for newly created files appears on the Configure | Preferences | Editing 1 dialog page

An option to set the file encoding format for newly created files appears on the Configure | Preferences | Editing 1 dialog page

Statistics

The statistics section displays the date and time when the current file was created/modified/accessed, as well the file size and file attributes. When a file attribute is set, it is displayed in normal density; when not set, it is displayed as grayed or disabled.

The Long Name and Short Name properties are displayed in read-only edit boxes to permit the strings to be copied to the Windows clipboard, if desired.

4.1.23 Toggle Read-Only

Menu: File > Toggle Read-Only

Default Shortcut Key: None

Macro function: ToggleReadOnly()

The Toggle Read-Only command can be used to toggle the status of the current file between read-only and writable. The current state is displayed in the status bar. 'WR' denotes that the file is writable. 'RO' indicates that the file is read-only.

This command duplicates the functionality available by double-clicking within the status bar in the read-only display panel, but it also provides the ability to assign that function to a key sequence (via Configure | Keyboard), if desired.

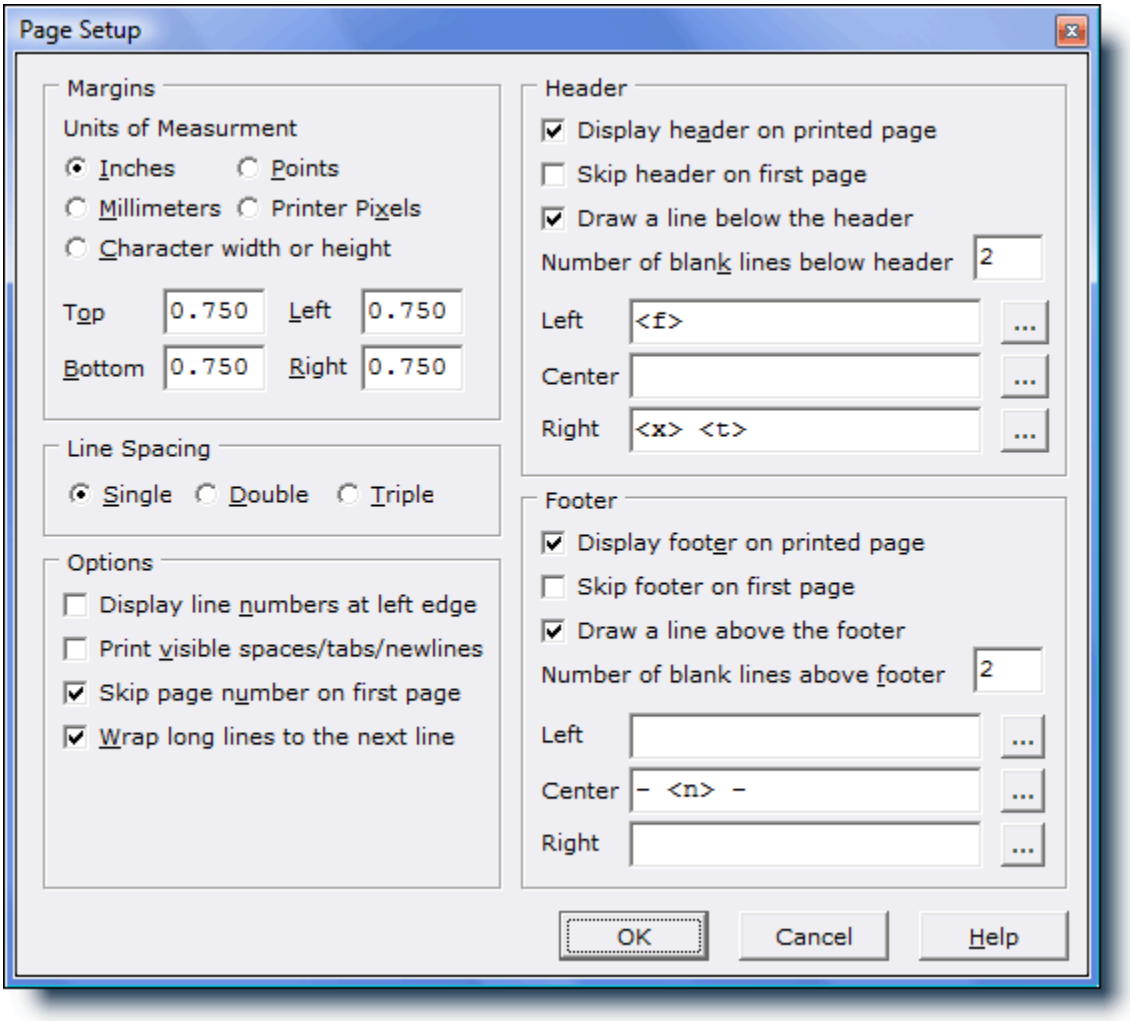
4.1.24 Page Setup

Menu: File > Page Setup

Default Shortcut Key: none

Macro function: PageSetup()

The Page Setup command provides access to a dialog box which controls the layout of the printed page. The following controls are provided:



☞ The settings made using the Page Setup command will be used for all print jobs that are performed from within Boxer. In other words, the Page Setup settings belong to the editor, and not to the current file. This method of operation differs from that of a Page Setup command within a word processor. A word processor is able to save its page settings within each document because the format of its documents is proprietary. As a text editor Boxer must save its files in ASCII format, and therefore cannot embed such information in the files it creates.

Margins

Inches
The margin values will be entered in inches. Use floating point values if needed (2.50), but not fractional values (2-1/2).

Millimeters
The margin values will be entered in millimeters.

Points
The margin values will be entered in points. There are 72 points to the inch.

Printer Pixels
The margin values will be entered in printer pixels. The number of pixels (dots) per inch will depend on the printer. Most printers are either 300 dpi or 600 dpi. This option provides the finest control over margin sizes.

Character width or height
The margin values are related to the width or height of a printed character. The Top and Bottom margin values will dictate the number of lines of margin. The Left and Right margin values will dictate the number of character widths of margin.

☞ Margin values can be no smaller than the printer's *non-printable zone*, which is the area at each edge of the paper on which the printer is physically incapable of printing. Boxer computes this value automatically, so you do not need to account for it when specifying margin values.

Top
Use this option to specify the distance from the top edge of the printed page to the *header* line, or to top of the body text when a header line is not used.

Bottom
Use this option to specify the distance from the bottom edge of the printed page to the *footer* line, or to the bottom of the body text when a footer line is not used.

Left
Use this option to specify the distance from the left edge of the paper to the beginning of the body text, or line numbers (if applicable).

Right
Use this option to specify the distance from the right edge of the paper to the right edge of the body text. Depending on the state of the *Wrap long lines to next line* checkbox (see below), long lines will either be wrapped to the next line or truncated.

☞ Changing the right margin value does not influence the wrap column of preformatted text as might occur within a Word Processor. Because Boxer is a Text Editor, not a Word Processor, the user alone controls hard line enders, and these are never re-wrapped without your knowledge. It may be necessary to experiment with different Printer Font sizes and/or different Text Width values to achieve optimum results on the printed page. The Print Preview command will be useful in this

regard, as it enables you to preview a print job on-screen without sending it to the printer.

Line Spacing

Select from **Single**, **Double** or **Triple**. Double and triple might be used when submitting a document which will be marked up by a teacher, for example.

Options

Display line numbers at left edge
If checked, line numbers will be applied at the left edge of the printed page. Overflow lines will be marked with a '+', and will therefore not alter the actual line count unnaturally.

Print visible spaces/tabs/newlines
If checked, Spaces, Tabs and Newline characters will be appear on the printed page as visible characters, using the same characters as have been configured for the Visible Spaces command.

The symbols which are used to represent Spaces, Tabs and Newlines are user-configurable. These can be set using options on the Configure | Preferences | Display options page. Separate options are provided for use with both ANSI and OEM printer fonts.

Skip page number on first page
If this option is selected, the page number will not be printed on the first page.

Wrap long lines to the next line
If checked, long lines will be wrapped to the next line rather than being truncated. Using a smaller Printer Font is one way to cure overflow lines. Another is to Reformat the document using a narrower Text Width.

Header

Display header on printed page
When checked, the related header controls become active and a *header* line will appear on all printed pages.

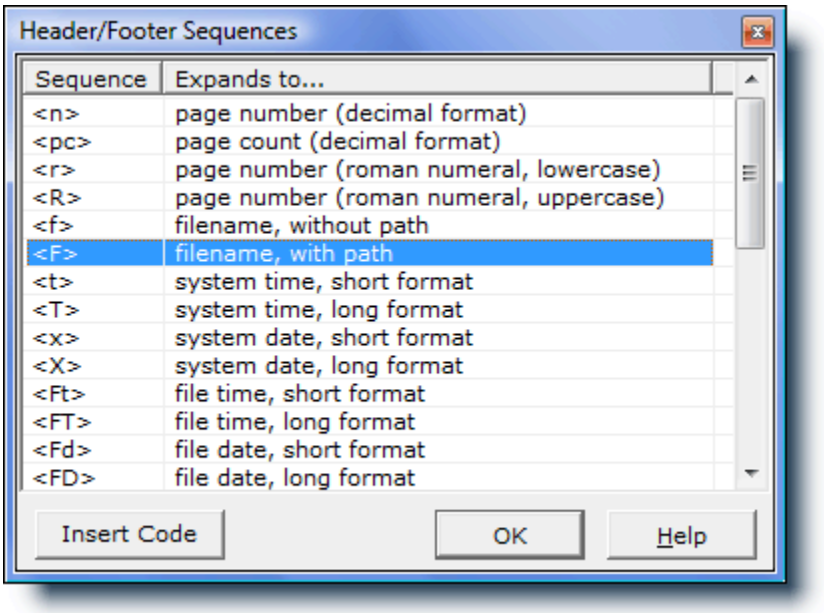
Skip header on first page
If checked, the header will not be printed on the first page.

Draw a line below the header
Use this option to cause a thin solid line to appear just the below the line of header text.

Number of blank lines below header
This option controls the number of lines of spacing between the header line and the top of the body text.

Left / Center / Right
These fields allow the text that is to appear in each header zone to be specified. You

may enter any text you like or use one or more of several pre-defined substitution sequences to insert a page number, filename, time, date, etc. Clicking the button to the right of each field displays the available sequences:



Footer

Display footer on printed page
When checked, the related footer controls become active and a *footer* line will appear on all printed pages.

Skip footer on first page
If checked, the footer will not be printed on the first page.

Draw a line above the footer
Use this option to cause a thin solid line to appear just above the line of footer text.

Number of blank lines above footer
This option controls the number of lines of spacing between the footer line and the bottom of the body text.

☞ In the evaluation version of Boxer, a 'watermark' will appear on all printed pages between the footer line and the bottom of the body text. This line serves as both a reminder and an encouragement to order a fully licensed copy. This reminder line is of course not present in the fully licensed version of Boxer.

Left / Center / Right
These fields allow the text which is to appear in each footer zone to be specified. You may enter any text you like or use one or more of several pre-defined substitution sequences to insert a page number, filename, time, date, etc. Click the button to the right of each field to select from the list of available sequences.

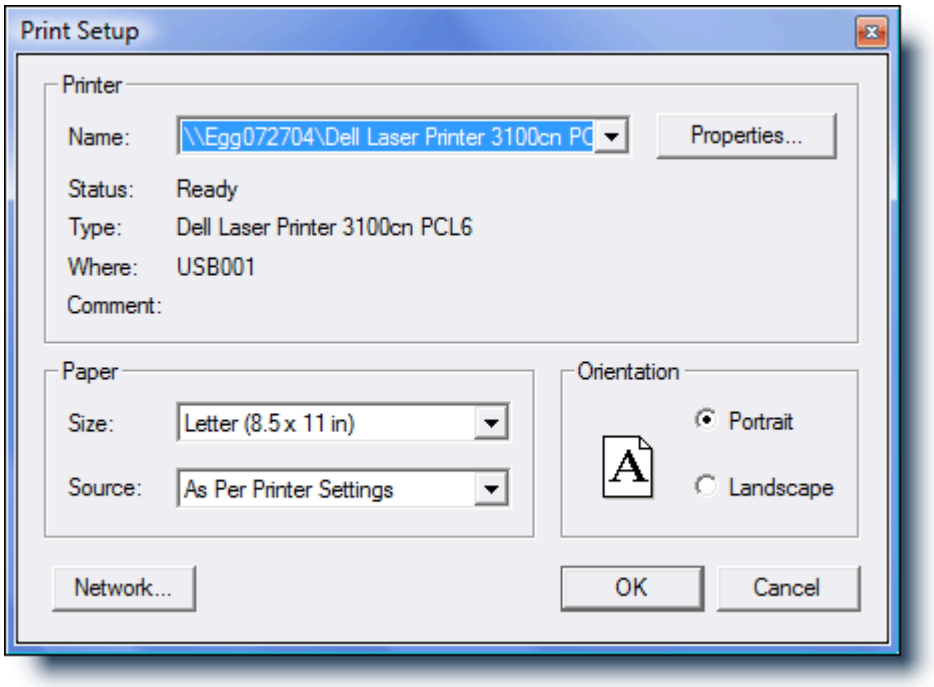
4.1.25 Print Setup

Menu: File > Print Setup

Default Shortcut Key: none

Macro function: PrintSetup()

The Print Setup command provides access to the standard Windows print setup dialog which allows the current printer to be selected. You can also select paper size, paper orientation (*portrait* or *landscape*), and other options which may vary from printer to printer.



4.1.26 Print Preview

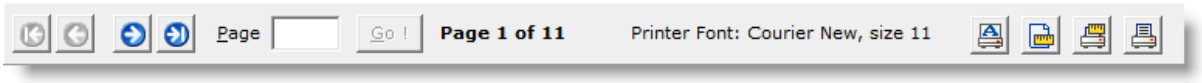
Menu: File > Print Preview > Normal / Color Syntax / Mono Syntax

Default Shortcut Key: none

Macro functions: PrintPreview, PrintPreviewMonochrome, PrintPreviewColor

The Print Preview command provides a means of viewing a print job on-screen before it is sent to the printer. The print document will be shown in a window that has controls for moving quickly to any page within the document. The *PgUp* and *PgDn* keys can also be used to page through the document. You can click with the left mouse in the turned page corner to advance to the next page; clicking with the right mouse button will move backward to the previous page.

The Print Preview form has buttons to access the Printer Font, Page Setup, Print Setup and Print commands directly from Print Preview mode. This makes it easy to see the effect of a font face or size change, or to view the result of changing margins, paper orientation, header and footer text, line spacing, line numbering or line wrapping.



Print Preview can be performed in any of three modes:

Normal
The preview is shown without the application of syntax highlighting coloration.

Monochrome Syntax
The preview is shown with monochrome syntax highlighting applied. Syntax elements will be displayed in bold, italic and/or underlined font styles in accordance with the current settings.

Color Syntax
The preview is shown with color syntax highlighting applied. Syntax elements will be displayed in color, and in bold, italic and/or underlined font styles in accordance with the current settings.

The Monochrome Syntax and Color Syntax preview modes are disabled unless the file being edited is a file type for which Syntax Highlighting information is defined.

The color and font style settings for Monochrome Syntax and Color Syntax are accessed from the Configure | Colors dialog by selecting the appropriate mode from the drop-down list at the upper left.

Because of the difference in resolution between the printer and the screen, the screen can never present a perfect image of the printed page. Under some circumstances, with certain font sizes, you may see imperfect line or character spacing or other small inaccuracies. It should not be assumed that the printed page will have the same inaccuracies. Print Preview can be expected to accurately show the layout of the page, page breaks, positioning, etc.

4.1.27 Print

Menu: File > Print > Print Normal / Print Color Syntax / Print Mono Syntax

Default Shortcut Key: Ctrl+P (Print Normal)

Macro functions: Print / PrintColor / PrintMonochrome

The Print command is used to send the current file to the printer. The layout of the printed page will be determined by the current settings within the Page Setup dialog. You may wish to use Print Preview to display the print job on-screen before sending it to the printer.

Printing can be performed in any of three modes:

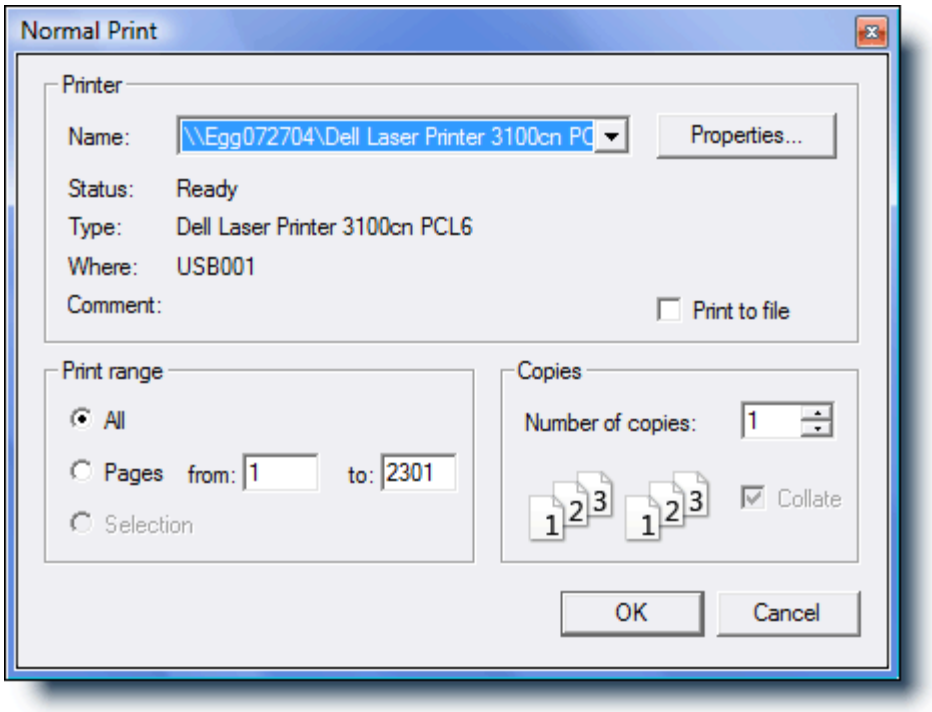
Normal
The file is printed without applying syntax highlighting coloration.

Color Syntax
The file is printed with color syntax highlighting applied. Syntax elements will be displayed in color, and in bold, italic and/or underlined font styles in accordance with the current settings.

Monochrome Syntax
The file is printed with monochrome syntax highlighting applied. Syntax elements will be displayed in bold, italic and/or underlined font styles in accordance with the current settings.

☞ The Monochrome Syntax and Color Syntax printing modes will be disabled when the file being edited is a file type for which Syntax Highlighting information is not available.

The standard Windows print dialog is presented before printing begins:



From here you can select the range of pages to be printed, and the number of copies to be printed. If text has been selected, you will be able to choose a radio button which dictates that only the selected text is to be printed. See the note below regarding printing selections of program source code.

Boxer will automatically convert any Tabs within the text being printed to Spaces. This

ensures that the Tab Display Size used by Boxer does not conflict with that of the printer.

The color/font settings for Monochrome Syntax Print and Color Syntax Print are accessed from the Configure | Colors dialog by selecting the appropriate mode from the drop-down list at the upper left.

☞ When printing selected text from program source code using either Monochrome or Color Syntax Print mode, improper highlighting can occur if the starting or ending points of the selection fall in the middle of a syntax element, or if the starting line of the selection falls within a multi-line comment block. Likewise, printing a columnar selection is likely to result in improper syntax highlighting. For best results, choose the selected area logically so that its start and end points occur at natural break points in the code.

☞ Strictly speaking, the use of Color Syntax Printing requires that a color printer be installed and attached. Some users may wish to use the *Print to File* option available from the Print dialog to save a print image on disk at one PC for later printing on another PC with a color printer. For this reason the Print Color Syntax command is not disabled when the PC is found to lack a color printer. Users will need to be careful when transporting a printer image file in this way, because it may not be compatible with the destination printer.

☞ A formfeed character can be placed in column one--or in the last position on a line--to indicate that a new page should begin at that point. The footer of the page--if one has been defined--will be printed and the page will eject. A formfeed in any other column will be ignored by Boxer's print routine.

4.1.28 Print All

Menu: File > Print > Print All Normal / Print All Color Syntax / Print All Mono Syntax

Default Shortcut Key: none

Macro functions: PrintAll / PrintAllColor / PrintAllMonochrome

The Print All command is used to send all files currently being edited to the printer. The layout of the printed page will be determined by the current settings within the Page Setup dialog. You may wish to use Print Preview to display the print jobs on-screen before sending it to the printer.

Printing can be performed in any of three modes:

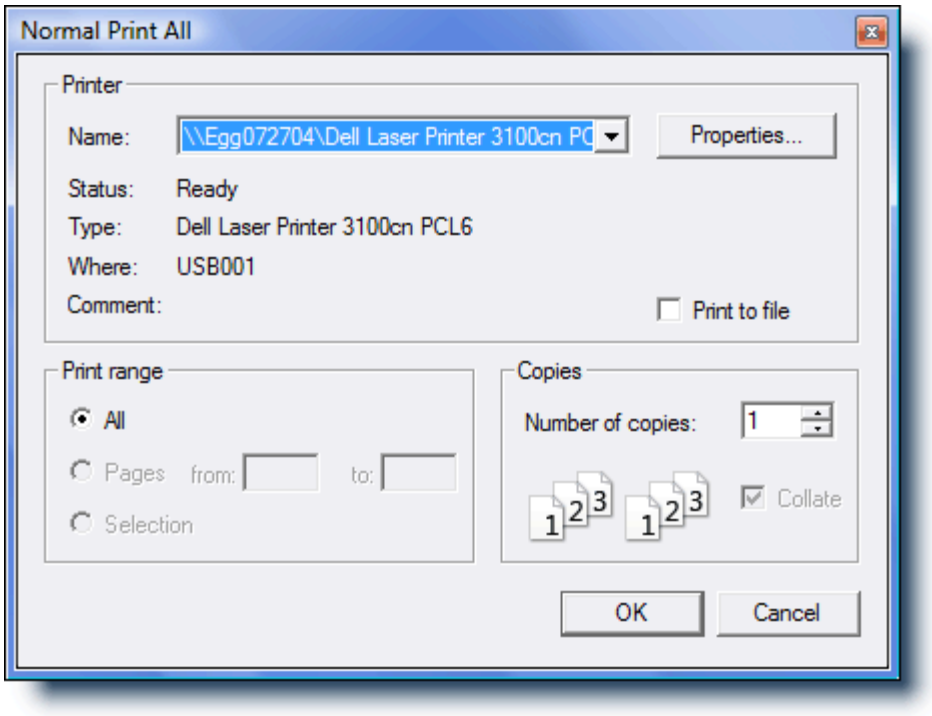
Normal
The file is printed without applying syntax highlighting coloration.

Color Syntax
The file is printed with color syntax highlighting applied. Syntax elements will be displayed in color, and in bold, italic and/or underlined font styles in accordance with the current settings.

Monochrome Syntax
The file is printed with monochrome syntax highlighting applied. Syntax elements will be displayed in bold, italic and/or underlined font styles in accordance with the current settings.

☞ The Monochrome Syntax and Color Syntax printing modes will be disabled when the file being edited is a file type for which Syntax Highlighting information is not available.

The standard Windows print dialog is presented before printing begins:



From here you can select the range of pages to be printed, and the number of copies to be printed. If text has been selected, you will be able to choose a radio button which dictates that only the selected text is to be printed. See the note below regarding printing selections of program source code.

Boxer will automatically convert any Tabs within the text being printed to Spaces. This ensures that the Tab Display Size used by Boxer does not conflict with that of the printer.

The color/font settings for Monochrome Syntax Print and Color Syntax Print are accessed from the Configure | Colors dialog by selecting the appropriate mode from the drop-down list at the upper left.

☞ When printing selected text from program source code using either Monochrome or Color Syntax Print mode, improper highlighting can occur if the starting or ending points of the selection fall in the middle of a syntax element, or if the starting line of the selection falls within a multi-line comment block. Likewise, printing a columnar

selection is likely to result in improper syntax highlighting. For best results, choose the selected area logically so that its start and end points occur at natural break points in the code.

☞ Strictly speaking, the use of Color Syntax Printing requires that a color printer be installed and attached. Some users may wish to use the *Print to File* option available from the Print dialog to save a print image on disk at one PC for later printing on another PC with a color printer. For this reason the Print Color Syntax command is not disabled when the PC is found to lack a color printer. Users will need to be careful when transporting a printer image file in this way, because it may not be compatible with the destination printer.

☞ A formfeed character can be placed in column one--or in the last position on a line--to indicate that a new page should begin at that point. The footer of the page--if one has been defined--will be printed and the page will eject. A formfeed in any other column will be ignored by Boxer's print routine.

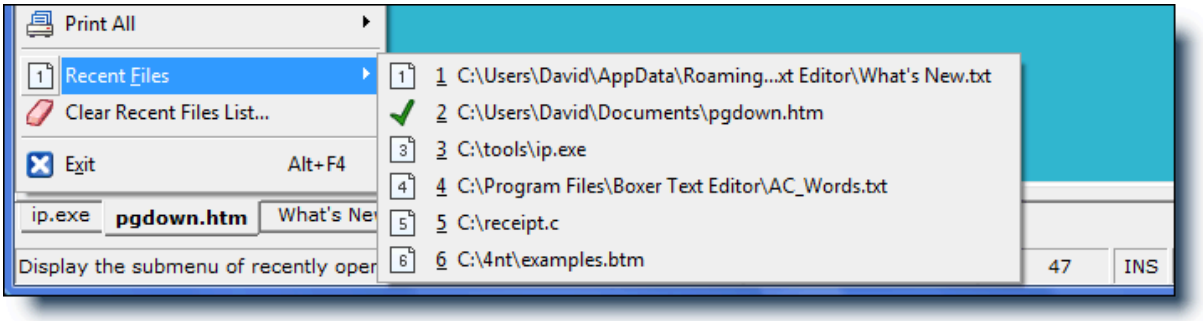
4.1.29 Recent Files

Menu: File

Default Shortcut Key: not applicable

Macro function: OpenRecentFile()

The Recent Files list appears near the bottom of the File menu, above the Exit command. Each time a file is opened for editing, its name is added to the list. If necessary, the eldest entry is bumped from the list. This list makes it easy to recall files which were recently viewed or edited without the need to use the Open command, as is typically done. The filenames are displayed with a 'hot' number to their left, so that *Alt+F*, *F*, followed by the number, will load the named file.



The number of files displayed in the list can be controlled on the Configure | Preferences | File I/O options page. The option is named *Number of recent files on the File menu*.

Long filenames will be shortened for display if the relevant option on the Configure | Preferences | Display options page is checked.

4.1.30 Clear Recent Files List

Menu: File > Clear Recent Files List

Default Shortcut Key: None

Macro function: ClearRecentFilesList()

Use this command to clear the record of recently accessed files from the Recent Files submenu.

4.1.31 Exit

Menu: File > Exit

Default Shortcut Key: Alt+F4

Macro function: Exit()

The Exit command is used to close Boxer and end your editing session. If unsaved changes have been made to any file a dialog box will appear for each such file to alert you to this condition. You will then be able to choose whether to save the changes before exiting, exit without saving, or cancel the Exit request altogether.

You can quickly tell whether a file has unsaved changes by looking for an asterisk (*) to the left of its name in the title bar, or on its File Tab.

Boxer has an option to warn on exit if the size of the text on the clipboard exceeds a user-defined threshold. This option can be found on the Configure | Preferences | Messages dialog.

4.2 Edit Menu

4.2.1 Undo

Menu: Edit > Undo

Default Shortcut Key: Ctrl+Z

Macro function: Undo()

The Undo command can be used to reverse the effect of the most recent change to the current file. Successive Undo commands will have the effect of stepping back in time, with each command undoing the previous change, until the limits of Undo become exhausted. If a change is undone which you'd like to get back, the Redo command can be used to 'undo undo'.

By default, cursor motion changes will also be undone. For example, if you're editing

mid-file and then jump to start of file to check something, Undo can be used to return you to your previous location. If you prefer that cursor motion commands *not* be stored for Undo, uncheck the relevant option on the Configure | Preferences | Editing 1 dialog page.

The Undo command does not affect the content of the current clipboard.

The size of the Undo buffer (in bytes) can be controlled on the Configure | Preferences | Editing 1 options page. The option is titled *Undo buffer size*. Values between 2048 and 65535 may be entered. The default value is 65535, which is also the maximum. There is little reason to select smaller values, as the memory cost is small compared to the utility that Undo provides.

Undo information is stored separately for each file, so there is no chance that excessive editing within one file can exhaust the undo capacity in another file.

An option is also provided to control whether or not Undo is allowed after the Save command. This option is titled *Allow Undo after File Save*

4.2.2 Undo All

Menu: Edit > Undo All

Default Shortcut Key: none

Macro function: UndoAll()

The Undo All command can be used to reverse the effect of all changes for which undo information is available. Unless you feel sure about the number of changes which have been recorded by Undo, it is often safer to use the Undo command to step singly through the changes so that their effect can be seen on-screen before proceeding. Should the Undo All command go 'too far', the Redo All command can be used to reverse its effect, or the Redo command can be used to restore the changes one at a time.

4.2.3 Redo

Menu: Edit > Redo

Default Shortcut Key: Ctrl+Y

Macro function: Redo()

The Redo command can be used to reverse the effect of the most recent Undo command. For example, while issuing a series of Undo commands to reverse unwanted changes, you suddenly see that the last Undo went too far. Issuing Redo will undo the last Undo. Undo and Redo are opposites.

Undo cannot be undone after additional changes are made; Redo must be issued before

other changes are made.

The Redo command is disabled until at least one Undo command has been performed.

4.2.4 Redo All

Menu: Edit > Redo All

Default Shortcut Key: none

Macro function: RedoAll()

The Redo All command can be used to undo the effect of all Undo commands which have been issued since the last change was made. Unless you feel sure about the number of undos which have been recorded by Redo, it is often safer to use the Redo command to step singly through the changes so that their effect can be seen on-screen before proceeding.

4.2.5 Clear Undo

Menu: Edit > Clear Undo

Default Shortcut Key: none

Macro function: ClearUndo()

The Clear Undo command can be used to clear the record of changes which are used by the Undo and Redo commands. After issuing this command, the record of changes for the current file is erased, and the Undo and Redo commands become disabled until additional changes are made.

4.2.6 Cut

Menu: Edit > Cut

Default Shortcut Key: Ctrl+X

Macro function: Cut()

The Cut command removes the selected text from the current file and places it on the current clipboard. The current clipboard might be the Windows clipboard or one of Boxer's eight internal clipboards. See the Edit | Set Clipboard command for details on changing the current clipboard.

If text is *not* selected, the Cut command will operate on the current line. This behavior can be controlled on the Configure | Preferences | Editing 1 options page. The option is titled *Cut/Copy/Append commands use current line when no text is selected*.

When a columnar selection (Block | Select Columnar) is placed on the clipboard, any under-length lines within the selected range will be extended with Spaces to match the width of the rectangular text block. This ensures that the block will behave as expected if the Paste command is later used to insert the text at a new location. Likewise, any Tab characters within the selected region will be converted to Spaces before being placed on the clipboard.

When operating in Typeover mode on a columnar selection, the Cut command will fill the selected area with Spaces without closing up the rectangle occupied by the text.

☞ When placing columnar text onto a clipboard, Boxer must take care so that subsequent Paste operations of that text will be performed properly. Columnar clipboard text must be pasted differently than stream text, since all lines must move rightward by the width of the text block. Notwithstanding this fact, columnar clipboard text placed onto the Windows clipboard by Boxer can still be pasted into other Windows applications. Boxer does not use a *private clipboard format* for this purpose.

☞ Boxer's clipboard commands will sense the type of text data being placed on the Windows clipboard and, when appropriate, use a more descriptive clipboard format to tag that data. For example, when Boxer senses that HTML code is being copied to the clipboard, the text will also be placed in the HTML clipboard format. This enables a conforming program to paste the data more intelligently. Boxer will also tag Rich Text data (RTF) and Comma-Separated Value (CSV).

4.2.7 Copy

Menu: Edit > Copy

Default Shortcut Key: Ctrl+C

Macro function: Copy()

The Copy command copies the selected text in the current file onto the current clipboard. The current clipboard might be the Windows clipboard or one of Boxer's eight internal clipboards. See the Edit | Set Clipboard command for details on changing the current clipboard.

If text is *not* selected, the Copy command will operate on the current line. This behavior can be controlled on the Configure | Preferences | Editing 1 options page. The option is titled *Cut/Copy/Append commands use current line when no text is selected*.

When a columnar selection (Block | Select Columnar) is placed on the clipboard, any under-length lines within the selected range will be extended with Spaces to match the width of the rectangular text block. This ensures that the block will behave as expected if the Paste command is later used to insert the text at a new location. Likewise, any Tab characters within the selected region will be converted to Spaces before being placed on the clipboard.

☞ Boxer's clipboard commands will sense the type of text data being placed on the Windows clipboard and, when appropriate, use a more descriptive clipboard format

to tag that data. For example, when Boxer senses that HTML code is being copied to the clipboard, the text will also be placed in the HTML clipboard format. This enables a conforming program to paste the data more intelligently. Boxer will also tag Rich Text data (RTF) and Comma-Separated Value (CSV).

4.2.8 Append

Menu: Edit > Append

Default Shortcut Key: Shift+Ctrl+C

Macro function: Append()

The Append command copies the selected text from the current file and adds it to the current clipboard. The current clipboard might be the Windows clipboard or one of Boxer's eight internal clipboards. See the Edit | Set Clipboard command for details on changing the current clipboard.

If text is *not* selected, the Append command will operate on the current line. This behavior can be controlled on the Configure | Preferences | Editing 1 options page. The option is titled *Cut/Copy/Append commands use current line when no text is selected*.

When a columnar selection (Block | Select Columnar) is placed on the clipboard, any under-length lines within the selected range will be extended with Spaces to match the width of the rectangular text block. This ensures that the block will behave as expected if the Paste command is later used to insert the text at a new location. Likewise, any Tab characters within the selected region will be converted to Spaces before being placed on the clipboard.

Text cannot be appended to a clipboard if the selection type (stream or columnar) differs from the type of the text which is already present on the clipboard.


When placing columnar text onto a clipboard, Boxer must take care so that subsequent Paste operations of that text will be performed properly. Columnar clipboard text must be pasted differently than stream text, since all lines must move rightward by the width of the text block. Notwithstanding this fact, columnar clipboard text placed onto the Windows clipboard by Boxer can still be pasted into other Windows applications. Boxer does not use a *private clipboard format* for this purpose.


Boxer's clipboard commands will sense the type of text data being placed on the Windows clipboard and, when appropriate, use a more descriptive clipboard format to tag that data. For example, when Boxer senses that HTML code is being copied to the clipboard, the text will also be placed in the HTML clipboard format. This allows a conforming program to paste the data more intelligently. Boxer will also tag Rich Text data (RTF) and Comma-Separated Value (CSV).

4.2.9 Cut Append

Menu: Edit > Cut Append

Default Shortcut Key: Shift+Ctrl+X

Macro function: CutAppend()

The Cut Append command removes the selected text from the current file and adds it to the current clipboard. The current clipboard might be the Windows clipboard or one of Boxer's eight internal clipboards. See the Edit | Set Clipboard command for details on changing the current clipboard.

If text is *not* selected, the Cut Append command will operate on the current line. This behavior can be controlled on the Configure | Preferences | Editing 1 options page. The option is titled *Cut/Copy/Append commands use current line when no text is selected*.

When a columnar selection (Block | Select Columnar) is placed on the clipboard, any under-length lines within the selected range will be extended with Spaces to match the width of the rectangular text block. This ensures that the block will behave as expected if the Paste command is later used to insert the text at a new location. Likewise, any Tab characters within the selected region will be converted to Spaces before being placed on the clipboard.

When operating in Typeover mode on a columnar selection, the Cut Append command will fill the selected area with Spaces without closing up the rectangle occupied by the text.

Text cannot be appended to a clipboard if the selection type (stream or columnar) differs from the type of the text which is already present on the clipboard.


When placing columnar text onto a clipboard, Boxer must take care so that subsequent Paste operations of that text will be performed properly. Columnar clipboard text must be pasted differently than stream text, since all lines must move rightward by the width of the text block. Notwithstanding this fact, columnar clipboard text placed onto the Windows clipboard by Boxer can still be pasted into other Windows applications. Boxer does not use a *private clipboard format* for this purpose.


Boxer's clipboard commands will sense the type of text data being placed on the Windows clipboard and, when appropriate, use a more descriptive clipboard format to tag that data. For example, when Boxer senses that HTML code is being copied to the clipboard, the text will also be placed in the HTML clipboard format. This enables a conforming program to paste the data more intelligently. Boxer will also tag Rich Text data (RTF) and Comma-Separated Value (CSV).

4.2.10 Paste

Menu: Edit > Paste

Default Shortcut Key: Ctrl+V

Macro function: Paste()

The Paste command inserts the text from the current clipboard at the location of the text cursor in the current file. The current clipboard might be the Windows clipboard or one of Boxer's eight internal clipboards. See the Edit | Set Clipboard command for details on changing the current clipboard.

If stream text (Edit | Select Stream) is being pasted, the clipboard text is inserted as if it had been typed from the keyboard. That is, the character at the text cursor is pushed along as far as is needed to accommodate the new text.

If columnar text (Edit | Select Columnar) is being pasted, the method of insertion is sensitive to the current edit mode. In Insert mode text will be pushed right to accommodate the size of the rectangular block being inserted. In Typeover mode the clipboard text will overwrite any text which may exist in the destination rectangle.

If an entire line has been placed on the clipboard by using the Copy or Cut command *without* first selecting text, the text cursor will be positioned to the start of line before the text is inserted.

The placement of the text cursor after a Paste operation can be controlled with an option on the Configure | Preferences | Editing 1 options page. The option is titled *Stay at insertion point when Pasting*.


If the Paste command is issued when no files are open, and when text is present on the active clipboard, a new file will be created automatically and the clipboard text will be pasted into that file.


When the Paste command is issued repeatedly to paste the same clipboard content, the status line will report a count of the number of times that the content has been pasted.


When placing columnar text onto a clipboard, Boxer must take care so that subsequent Paste operations of that text will be performed properly. Columnar clipboard text must be pasted differently than stream text, since all lines must move rightward by the width of the text block. Notwithstanding this fact, columnar clipboard text placed onto the Windows clipboard by Boxer can still be pasted into other Windows applications. Boxer does not use a *private clipboard format* for this purpose.

4.2.11 Paste As

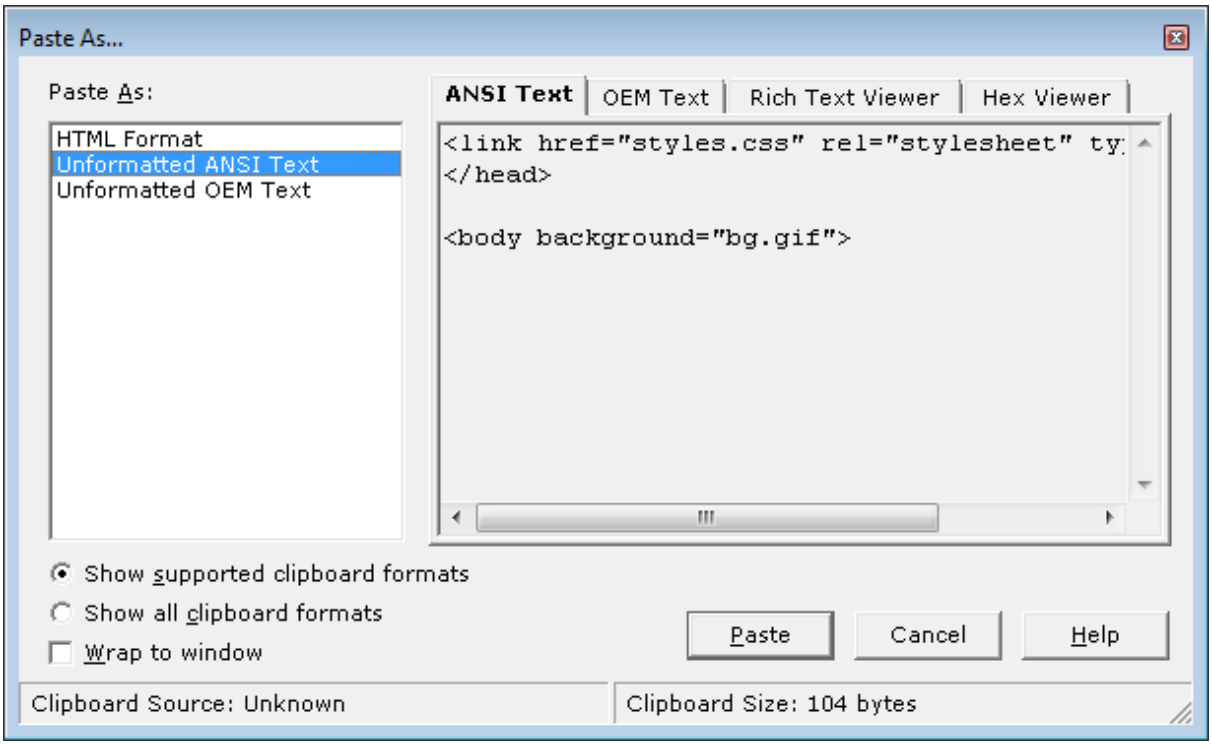
Menu: Edit > Paste As

Default Shortcut Key: Shift+Ctrl+V

Macro function: PasteAs()

The Paste As command allows the content of the Windows clipboard to be viewed in various formats, and with various viewers. This makes it possible to Paste text into Boxer in formats other than normal text. After selecting the desired format, simply click the *Paste* button.

Background: the Windows clipboard is capable of holding multiple pieces of data simultaneously. Sometimes this capability is exploited so that different versions of the same data can be made available. In other cases, different formats will hold more descriptive copies of the same data. For example, when you copy text from a web page that is being displayed in Internet Explorer, the data will be placed on the clipboard in several different formats. These formats are HTML Format, Rich Text Format (RTF), ANSI Text and OEM Text. When another application pastes that data from the clipboard, it chooses the format that is most meaningful to that application. Boxer's Paste command would use the ANSI Text format. This is where the Paste As command becomes useful. There might be times when you would prefer that Boxer be able to paste the clipboard data in HTML format, so that HTML formatting codes and hyperlinks are not lost. For some users, having access to the data in RTF format might prove useful. The Paste As command allows all available formats to be viewed so that the most useful format can be used.



Boxer's clipboard commands will sense the type of text data being placed on the Windows clipboard and, when appropriate, use a more descriptive clipboard format to tag that data. For example, when Boxer senses that HTML code is being copied to the clipboard, the text will also be placed in the HTML clipboard format. This enables a conforming program to paste the data more intelligently. Boxer will also tag Rich Text data (RTF) and Comma-Separated Value (CSV).

4.2.12 Delete

Menu: Edit > Delete

Default Shortcut Key: Del

Macro function: Delete()

The Delete command deletes the selected text from the current file. If text is not selected, the character at the text cursor is deleted.

When operating in Typeover mode on a columnar selection, the Delete command will fill the selected area with Spaces without closing up the rectangle occupied by the text.

4.2.13 Select All Text

Menu: Edit > Select All Text

Default Shortcut Key: Ctrl+A

Macro function: SelectAllText()

The Select All Text command can be used to quickly select all text within the current file. The selected text can then be operated upon in all the ways in which manually selected text might be manipulated.

If the default selection mode is columnar (Edit | Select Columnar), the selection mode must be changed to stream mode (Edit | Select Stream) in order to perform the Select All Text command.

4.2.14 Copy Filename

Menu: Edit > Copy Filename

Default Shortcut Key: none

Macro function: CopyFilename()

The Copy Filename command copies the full filepath of the current window to the current clipboard.

To insert the filepath of the current file into the edited text, use the Insert Filename command.

This command can be useful when supplying the name of an edited file to an email program for attachment to a message.

4.2.15 Paste Clipboard

Menu: Edit > Paste Clipboard > Clipboard *n*

Default Shortcut Key: Shift+Alt+*n*

Macro function: PasteClipboard()

The Paste Clipboard commands permit the content of any clipboard to be inserted into the text file directly, without the need to first select that clipboard with the Set Clipboard command before using Paste.

The content of Boxer's internal clipboards is saved from session to session (subject to a limit; see Sizes and Limits), ensuring that their content is always available. Since the Paste Clipboard commands allow pasting from any clipboard with a single key sequence, the internal clipboards can be the ideal place to store commonly used text blocks.

When the content of a clipboard is displayed in a popup window, the text is displayed with an 8 point, fixed width, *Courier New* font. This font utilizes the ANSI character set mapping. If the current screen font uses an OEM character set mapping, and if characters outside the normal alphanumeric range reside on the clipboard, then the content of the clipboard may appear different in the popup window than it would in the underlying file. This difference is simply the result of a difference in character sets, and does not mean that the data on the clipboard has been adjusted or corrupted.

4.2.16 Set Clipboard

Menu: Edit > Set Clipboard > Clipboard *n*

Default Shortcut Key: none

Macro function: SetClipboard()

The Set Clipboard command is used to set the active clipboard. The active clipboard can be either the Windows clipboard or one of Boxer's eight internal (private) clipboards. Text which is placed on the Windows clipboard will be accessible by other applications. Likewise, text placed on the Windows clipboard by other applications is available to Boxer whenever the active clipboard is the Windows clipboard. Text that is placed on any of Boxer's internal clipboards is *not* available to other applications.

The content of each clipboard is displayed in a popup window as the menu cursor is moved across the clipboard's menu entry. This makes it easy to check what's on a clipboard without actually pasting the content into a file.

The content of Boxer's internal clipboards will be saved at the end of an edit session, as long as the length of the text on the clipboard is 2,048 characters or less. Because the content of the internal clipboards persists from session to session, and cannot be changed by other applications, these clipboards can be useful for storing frequently used text blocks for insertion into your files. The Edit Clipboard command might be used to create these text blocks and maintain them.

The content of a single clipboard can be cleared with the Clear Clipboard command. The content of *all* clipboards can be cleared with the Clear All Clipboards command.

When the content of a clipboard is displayed in a popup window, the text is

displayed with an 8 point, fixed width, *Courier New* font. This font utilizes the ANSI character set mapping. If the current screen font uses an OEM character set mapping, and if characters outside the normal alphanumeric range reside on the clipboard, then the content of the clipboard may appear different in the popup window than it would in the underlying file. This difference is simply the result of a difference in character sets, and does not mean that the data on the clipboard has been adjusted or corrupted.

4.2.17 Set Clipboard -> Previous

Menu: Edit > Set Clipboard > Previous

Default Shortcut Key: none

Macro function: SetClipboardPrevious()

This command can be used to set the active clipboard to be the previous clipboard. For example, if clipboard 3 is active, issuing this command will make clipboard 2 the active clipboard. Clipboard 8 is considered the previous clipboard to the Windows clipboard.

4.2.18 Set Clipboard -> Next

Menu: Edit > Set Clipboard > Next

Default Shortcut Key: none

Macro function: SetClipboardNext()

This command can be used to set the active clipboard to be the next clipboard. For example, if clipboard 3 is active, issuing this command will make clipboard 4 the active clipboard. The Windows clipboard is considered the next clipboard to the clipboard 8.

4.2.19 Edit Clipboard

Menu: Edit > Edit Clipboard

Default Shortcut Key: none

Macro function: EditClipboard()

The Edit Clipboard command can be used to edit the content of the clipboard selected. The content of the clipboard is placed into an editing window and can be edited in all the same ways a file may be edited. When the text in the window is saved, it is written back to the clipboard. The Windows clipboard and any of Boxer's internal clipboards can be edited. The Windows clipboard is not eligible for editing if it contains non-text data.

The content of each clipboard is displayed in a popup window as the menu cursor is moved across the clipboard's menu entry. This makes it easy to check what's on a

clipboard without pasting the content into a file or opening it for editing.

The content of Boxer's internal clipboards will be saved at the end of an edit session, as long as the length of the text on the clipboard is 2,048 characters or less. Because the content of the internal clipboards persists from session to session, and cannot be changed by other applications, these clipboards can be useful for storing frequently used text blocks for insertion into your files. The Edit Clipboard command might be used to create these text blocks and maintain them.

The content of a clipboard can be cleared with the Clear Clipboard command. The content of *all* clipboards can be cleared with the Clear All Clipboards command.


When the content of a clipboard is displayed in a popup window, the text is displayed with an 8 point, fixed width, *Courier New* font. This font utilizes the ANSI character set mapping. If the current screen font uses an OEM character set mapping, and if characters outside the normal alphanumeric range reside on the clipboard, then the content of the clipboard may appear different in the popup window than it would in the underlying file. This difference is simply the result of a difference in character sets, and does not mean that the data on the clipboard has been adjusted or corrupted.

4.2.20 Clear Clipboard

Menu: Edit > Clear Clipboard > Clipboard *n*

Default Shortcut Key: none

Macro function: ClearClipboard()

The Clear Clipboard command can be used to clear (erase) the content of the clipboard selected. The content of each clipboard is displayed in a popup window as the menu cursor is moved across the clipboard's menu entry. This makes it easy to check what's on a clipboard before deciding whether to clear it.

The effect of the Clear Clipboard command cannot be undone with Undo, so use this command carefully.


When the content of a clipboard is displayed in a popup window, the text is displayed with an 8 point, fixed width, *Courier New* font. This font utilizes the ANSI character set mapping. If the current screen font uses an OEM character set mapping, and if characters outside the normal alphanumeric range reside on the clipboard, then the content of the clipboard may appear different in the popup window than it would in the underlying file. This difference is simply the result of a difference in character sets, and does not mean that the data on the clipboard has been adjusted or corrupted.


The Clear Windows Clipboard command will remain enabled even when the Windows Clipboard contains non-text data. This allows the content of the clipboard to be cleared by Boxer, in case this operation is desired.

4.2.21 Clear Clipboard -> All Clipboards

Menu: Edit > Clear Clipboard > All Clipboards

Default Shortcut Key: none

Macro function: ClearAllClipboards()

The Clear All Clipboards command can be used to clear (erase) the content of all clipboards. The Windows clipboard and the eight internal clipboards will all be affected.

The effect of the Clear All Clipboards command cannot be undone with Undo, so use this command carefully.

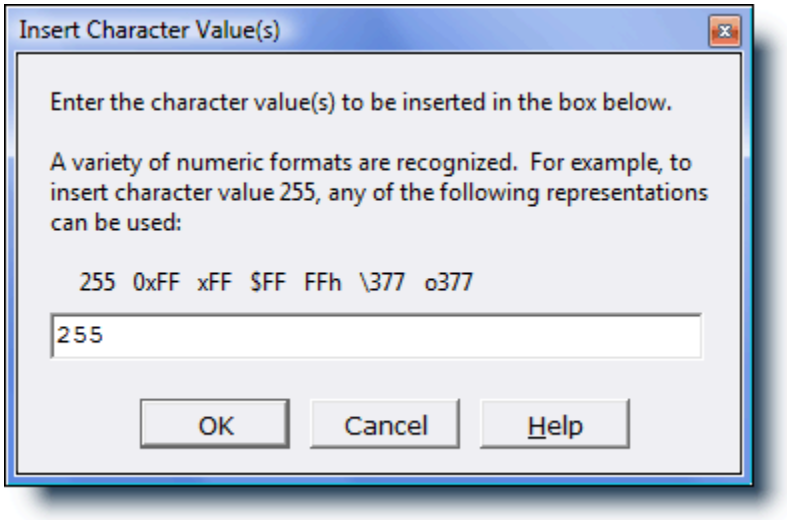
4.2.22 Insert -> Character(s)

Menu: Edit > Insert > Character(s)

Default Shortcut Key: none

Macro function: InsertCharacter()

The Insert Character(s) command can be used to insert one or more characters by specifying their numeric values. The values to be entered are typed into a popup dialog box. This command is useful for entering characters which are not readily typed from the keyboard, such as those values below the Space (character value 32), and those above 127.



The ANSI Chart and OEM Chart can also be used to insert non-standard characters into a file. After locating the desired character in the chart, simply press *Enter* or double click on the selected entry.

When the need to insert a special character or symbol arises frequently, consider using

the Insert Symbols feature rather than the Insert Character command. The Insert Symbols feature permits a defined character to be entered using a single keystroke.

For additional information, see the Inserting Special Characters topic.

Boxer's Value at Cursor command can be used to verify the value of the character at the cursor.

On most PCs, a character can be entered from the keyboard by typing its numeric value in a special way. With the *Numlock* key on, depress and hold the *Alt* key. Then type the 0 (zero) on the numeric keypad, followed by the *decimal* value of the character to be inserted. Finally, release the *Alt* key. The character whose value was typed will appear at the text cursor.

4.2.23 Insert -> Formfeed

Menu: Edit > Insert > Formfeed

Default Shortcut Key: none

Macro function: Formfeed()

The Insert Formfeed command can be used to quickly insert the formfeed, character value 12, at the current text cursor location. The formfeed character is recognized by printers as a request to eject the current page and advance to (or load) a new page.

When printing a text file from within Boxer, a formfeed character can be placed in column one--or in the last position on a line--to indicate that a new page should begin at that point. The footer of the page--if one has been defined--will be printed and the page will eject. A formfeed in any other column will be ignored by Boxer's printing service.

4.2.24 Insert -> Tab

Menu: Edit > Insert > Tab

Default Shortcut Key: Tab

Macro function: Tab()

The Insert Tab command inserts a Tab (character value 9) at the text cursor location. After insertion, the text cursor moves to the next tabstop, as determined by the Tab Display Size.

If the Tab key has been configured to insert Spaces, an equivalent number of Spaces will be inserted instead of a Tab. The option which controls this behavior appears on the Configure | Preferences | Tabs options page. The option is titled *Tab key inserts spaces*.

The Configure | Preferences | Tabs page also contains an option for the Tab key to

insert Spaces and obtain its tabstops from the line above. When this option is used, the *Tab* key will advance the text cursor to the next field of data as determined from the line above the current line. This option is especially useful when editing tabular data within a table or chart.

Boxer's default Tab Display Size is 4, which permits program source code with several indent levels to be displayed without exceeding the screen width. Many other programs, and most printers, will treat Tabs as having a display size of 8. You may need to make adjustments in order to print or display files with another program which does not use a Tab display size of 4. One remedy could be to use the Tabs to Spaces command to convert a copy of the file before using it with the other program. Note that Boxer's Print command will automatically convert Tabs to Spaces before sending its data to the printer, so there will be no such difficulty when printing files from within Boxer.

Tabs, Spaces and Newline characters can be made visible with the Visible Spaces command.


If the Insert Tab command is issued when a range of lines is selected, the Indent one Tabstop command will be performed. If a small selection is present on a single line the selection will be replaced with a Tab character.

4.2.25 Insert -> Filename

Menu: Edit > Insert > Filename

Default Shortcut Key: Tab

Macro function: InsertFilename()

The Insert Filename command inserts the full filepath of the current window into the edited text.

To copy the filepath of the current window to the clipboard, use the Copy Filename command.


When editing source code, use this command to quickly place the name of the file into a program comment.

4.2.26 Insert -> HTML Image Tag

Menu: Edit > Insert > HTML Image Tag

Default Shortcut Key: none

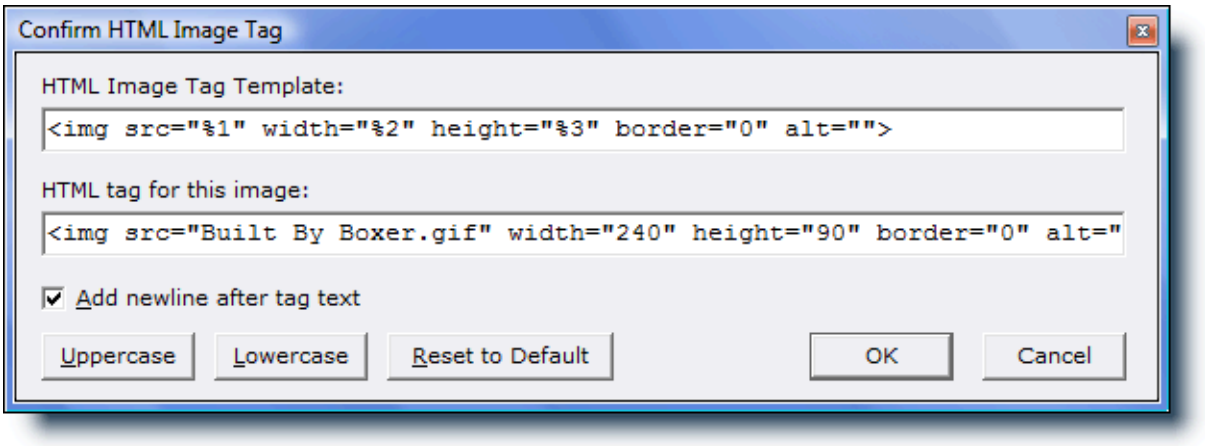
Macro function: HTMLImageTag()

The Insert HTML Image Tag command can be used to insert an HTML image tag into the current file for a selected graphics file. The image tag will use the filename, image height, and image width of the selected image file. The following image file formats are

supported: BMP, GIF and JPEG.

A dialog will appear so that the name of the image file can be selected. If you like, multiple image files can be selected at the same time.

Before the image tag is created, a dialog appears to confirm the operation, and to provide access to the image template:



You can control the format of the image tag by editing the *HTML Image Tag Template* in the upper edit box. The format of the template string can be changed freely, so long as the `%1`, `%2` and `%3` sequences appear in the string, and remain associated with the filename (`src`), `width` and `height` properties, respectively. The image tag that will be inserted appears in the lower edit box. Buttons are provided to quickly convert the tag to uppercase or lowercase, as well as a *Reset to Default* button that will restore the template string to its original form.

You can also activate the Insert HTML Image Tag feature by dragging and dropping images onto Boxer.

4.2.27 Insert -> Line Below

Menu: Edit > Insert > Line Below

Default Shortcut Key: Ctrl+Enter

Macro function: InsertLineBelow()

The Insert Line Below command can be used to create a new line beneath the current line. The effect of this command is the same as moving the cursor to the end of line and pressing *Enter* while in Insert mode.

4.2.28 Insert -> Line Above

Menu: Edit > Insert > Line Above

Default Shortcut Key: Shift+Ctrl+Enter

Macro function: InsertLineAbove()

The Insert Line Above command can be used to create a new line above the current line. The effect of this command is the same as moving the cursor to the end of the previous line and pressing *Enter*, while in Insert mode.

4.2.29 Insert -> Short Date

Menu: Edit > Insert > Short Date

Default Shortcut Key: Shift+F11

Macro function: InsertShortDate()

The Insert Short Date command can be used to insert a text string representing the current date, in *short date* format. A preview of the string which will be inserted is displayed on the Status Bar when the menu cursor is moved onto the Short Date menu entry.

The format used to display the short date is in accordance with the regional settings for date display as defined on your computer. To change these settings, see Start Menu | Settings | Control Panel | Regional Settings | Date.

Double clicking atop the date display on the Status Bar will also issue the Insert Short Date command.

4.2.30 Insert -> Long Date

Menu: Edit > Insert > Long Date

Default Shortcut Key: Shift+Ctrl+F11

Macro function: InsertLongDate()

The Insert Long Date command can be used to insert a text string representing the current date, in *long date* format. A preview of the string which will be inserted is displayed on the Status Bar when the menu cursor is moved onto the Long Date menu entry.

The format used to display the long date is in accordance with the regional settings for date display as defined on your computer. To change these settings, see Start Menu | Settings | Control Panel | Regional Settings | Date.

4.2.31 Insert -> Short Time

Menu: Edit > Insert > Short Time

Default Shortcut Key: Shift+F12

Macro function: InsertShortTime()

The Insert Short Time command can be used to insert a text string representing the current time, in *short time* format. A preview of the string which will be inserted is displayed on the Status Bar when the menu cursor is moved onto the Short Time menu entry.

The format used to display the short time is in accordance with the regional settings for time display as defined on your computer. To change these settings, see Start Menu | Settings | Control Panel | Regional Settings | Time.

Double clicking atop the time display on the Status Bar will also issue the Insert Short Time command.

4.2.32 Insert -> Long Time

Menu: Edit > Insert > Long Time

Default Shortcut Key: Shift+Ctrl+F12

Macro function: InsertLongTime()

The Insert Long Time command can be used to insert a text string representing the current time, in *long time* format. A preview of the string which will be inserted is displayed on the Status Bar when the menu cursor is moved onto the Long Time menu entry.

The format used to display the long time is in accordance with the regional settings for time display as defined on your computer. To change these settings, see Start Menu | Settings | Control Panel | Regional Settings | Time.

4.2.33 Delete -> Previous Word

Menu: Edit > Delete > Previous Word

Default Shortcut Key: Ctrl+Backspace

Macro function: DeletePreviousWord()

The Delete Previous Word command deletes from the text cursor to the end of the previous word.

The characters which serve to delimit words can be set on the Configure | Preferences | Cursor options page. The option is titled *These characters will delimit words.*

4.2.34 Delete -> Next Word

Menu: Edit > Delete > Next Word

Default Shortcut Key: Ctrl+Del

Macro function: DeleteNextWord()

The Delete Next Word command deletes from the text cursor to the beginning of the next word.

The characters which serve to delimit words can be set on the Configure | Preferences | Cursor options page. The option is titled *These characters will delimit words.*

4.2.35 Delete -> Current Line

Menu: Edit > Delete > Current Line

Default Shortcut Key: Alt+D

Macro function: DeleteLine()

The Delete Current Line command deletes all of the text and the newline character on the current line. Whenever possible, the column of the text cursor will be preserved when moving to the next line.

4.2.36 Delete -> to End of Line

Menu: Edit > Delete > to End of Line

Default Shortcut Key: none

Macro function: DeleteToEndOfLine()

The Delete to End of Line command deletes from the text cursor to the end of line. The Newline character is not deleted. The position of the text cursor is unchanged.

If issued on an empty line with the text cursor in column 1, the Delete to End of Line command will delete the entire line.

4.2.37 Delete -> to Start of Line

Menu: Edit > Delete > to Start of Line

Default Shortcut Key: Ctrl+K

Macro function: DeleteToStartOfLine()

The Delete to Start of Line command deletes from the left of the text cursor to the start of line. The cursor is left in Column 1.

If issued on an empty line with the text cursor in column 1, the Delete to Start of Line command will delete the entire line.

4.2.38 Delete -> Lines that Begin with

Menu: Edit > Delete > Lines That Begin With

Default Shortcut Key: none

Macro function: DeleteLinesThatBeginWith()

This command can be used to delete all lines that begin with a user-specified text string. If a range of lines is selected, the operation will be restricted to the selected range.

The text string is supplied in a dialog box that appears after the command is issued. The "Match case" checkbox on that dialog can be used to control how the search is performed: with or without case sensitivity. Before the operation is performed, a second dialog box appears which tells the number of lines that will be deleted, and provides an opportunity to cancel the operation.

4.2.39 Delete -> Lines that End with

Menu: Edit > Delete > Lines That End With

Default Shortcut Key: none

Macro function: DeleteLinesThatEndWith()

This command can be used to delete all lines that end with a user-specified text string. If a range of lines is selected, the operation will be restricted to the selected range.

The text string is supplied in a dialog box that appears after the command is issued. The "Match case" checkbox on that dialog can be used to control how the search is performed: with or without case sensitivity. Before the operation is performed, a second dialog box appears which tells the number of lines that will be deleted, and provides an opportunity to cancel the operation.

4.2.40 Delete -> Lines that Contain

Menu: Edit > Delete > Lines That Contain

Default Shortcut Key: none

Macro function: DeleteLinesThatContain()

This command can be used to delete all lines that contain a user-specified text string. If a range of lines is selected, the operation will be restricted to the selected range.

The text string is supplied in a dialog box that appears after the command is issued. The "Match case" checkbox on that dialog can be used to control how the search is performed: with or without case sensitivity. Before the operation is performed, a second dialog box appears which tells the number of lines that will be deleted, and provides an opportunity to cancel the operation.

4.2.41 Delete -> Lines that do not Begin with

Menu: Edit > Delete > Lines That Do Not Begin With

Default Shortcut Key: none

Macro function: DeleteLinesThatDoNotBeginWith()

This command can be used to delete all lines that **do not** begin with a user-specified text string. If a range of lines is selected, the operation will be restricted to the selected range.

The text string is supplied in a dialog box that appears after the command is issued. The "Match case" checkbox on that dialog can be used to control how the search is performed: with or without case sensitivity. Before the operation is performed, a second dialog box appears which tells the number of lines that will be deleted, and provides an opportunity to cancel the operation.

4.2.42 Delete -> Lines that do not End with

Menu: Edit > Delete > Lines That Do Not End With

Default Shortcut Key: none

Macro function: DeleteLinesThatDoNotEndWith()

This command can be used to delete all lines that **do not** end with a user-specified text string. If a range of lines is selected, the operation will be restricted to the selected range.

The text string is supplied in a dialog box that appears after the command is issued. The "Match case" checkbox on that dialog can be used to control how the search is performed: with or without case sensitivity. Before the operation is performed, a second dialog box appears which tells the number of lines that will be deleted, and provides an opportunity to cancel the operation.

4.2.43 Delete -> Lines that do not Contain

Menu: Edit > Delete > Lines That Do Not Contain

Default Shortcut Key: none

Macro function: DeleteLinesThatDoNotContain()

This command can be used to delete all lines that **do not** contain a user-specified text string. If a range of lines is selected, the operation will be restricted to the selected range.

The text string is supplied in a dialog box that appears after the command is issued. The "Match case" checkbox on that dialog can be used to control how the search is performed: with or without case sensitivity. Before the operation is performed, a second dialog box appears which tells the number of lines that will be deleted, and provides an opportunity to cancel the operation.

4.2.44 Delete -> Blank Lines

Menu: Edit > Delete > Blank Lines

Default Shortcut Key: none

Macro function: DeleteBlankLines()

The Delete Blank Lines command can be used to delete blank lines within the current file. If a range of lines is selected, the operation will be restricted to the selected lines. Due to the destructive nature of this command, a confirmation is required before the operation begins.

☞ A line is considered blank if it has no text, or if its text consists only of *whitespace*.

4.2.45 Delete -> Duplicate Lines

Menu: Edit > Delete > Duplicate Lines

Default Shortcut Key: none

Macro function: DeleteDuplicateLines()

The Delete Duplicate Lines command can be used to delete duplicate lines within the current file. If a range of lines is selected, the operation will be restricted to the selected lines. Due to the destructive nature of this command, a confirmation is required before the operation begins.

Delete Duplicate Lines will not delete the first instance of a duplicated line. In other words, given a file that contained five lines with the text '`sample`', four of these lines would be deleted.

This command is similar in effect to the Find Unique Lines and Find Distinct Lines commands. For certain tasks, one of these commands might be more suitable.

To delete blank lines, use the Delete Blank Lines command.

4.2.46 Delete -> Bookmarked Lines

Menu: Edit > Delete > Bookmarked Lines

Default Shortcut Key: none

Macro function: DeleteBookmarkedLines()

The Delete Bookmarked Lines command can be used to delete all bookmarked lines from the current file. For example, the Toggle Bookmark command can be used to 'flag' several lines for deletion, and then the Delete Bookmarked Lines command can be used to delete the lines.

This command deletes lines, not simply bookmarks. To remove the bookmarks from bookmarked lines, use either the Toggle Bookmark or the Bookmark Manager command. If you have accidentally deleted bookmarked lines when you meant only to clear their bookmarks, use the Undo command to recover these lines.

4.2.47 Line -> Duplicate Line

Menu: Edit > Line > Duplicate Line

Default Shortcut Key: F2

Macro function: DuplicateLine()

The Duplicate Line command can be used to make a copy of the current line. The new line will be created below the current line, and the text cursor will be moved onto the new line at the current cursor column.

The Duplicate Line command is a quick alternative to using the Copy and Paste commands to perform the same task, and does not affect the content of the current clipboard.

When the Duplicate Line command is issued repeatedly to duplicate a line, the status line will report a count of the number of times the command has issued.

4.2.48 Line -> Duplicate and Increment

Menu: Edit > Line > Duplicate and Increment

Default Shortcut Key: Shift+F2

Macro function: DuplicateAndIncrement()

The Duplicate and Increment command is similar to the Duplicate Line command, with one important difference: as it copies the current line to a new line below, it increments any values it finds within the line. A few examples will help illustrate its utility:

When the cursor is placed on a line with the following text:

```
width1 = MainForm->WidthArray[1];
```

and the Duplicate and Increment command is issued three times, the following text will result:

```
width1 = MainForm->WidthArray[1];
width2 = MainForm->WidthArray[2];
width3 = MainForm->WidthArray[3];
width4 = MainForm->WidthArray[4];
```

Duplicate and increment also recognizes character constants...

```
char01 = 'A';
```

would become:

```
char01 = 'A';
char02 = 'B';
char03 = 'C';
char04 = 'D';
```

... and on hexadecimal values:

```
pos[15] := $DF;
```

becomes:

```
pos[15] := $DF;
pos[16] := $E0;
pos[17] := $E1;
pos[18] := $E2;
```

Hexadecimal values are recognized in three forms: 0xFF, FFh and $FF.

The examples above relate to programming, but Duplicate and Increment can also be useful in non-technical situations. If you needed to start a numbered list of items, you could create the first line:

```
Part No. 3141001
```

and then use Duplicate and Increment to make as many copies as needed:

```
Part No. 3141001
Part No. 3141002
Part No. 3141003
Part No. 3141004
Part No. 3141005
```

When the Duplicate and Increment command is issued repeatedly to duplicate a line, the status line will report a count of the number of times the command has issued.

4.2.49 Line -> Move Line Up

Menu: Edit > Line > Move Line Up

Default Shortcut Key: Shift+Alt+Up

Macro function: MoveLineUp()

The Move Line Up command moves the text of the current line to the line above, and moves the text cursor to that line so it follows the line being moved. This command can be useful when rearranging items in an ordered list, since it removes the need to select, cut and then paste text from the clipboard as a means of reordering a series of lines.

If the current line is the first line in the file, the command has no effect.

See also: Move Line Down

4.2.50 Line -> Move Line Down

Menu: Edit > Line > Move Line Down

Default Shortcut Key: Shift+Alt+Down

Macro function: MoveLineDown()

The Move Line Down command moves the text of the current line to the line below, and moves the text cursor to that line so it follows the line being moved. This command can be useful when rearranging items in an ordered list, since it removes the need to select, cut and then paste text from the clipboard as a means of reordering a series of lines.

If the current line is the last line in the file, the command has no effect.

The effect of this command is similar to that of the Swap Lines command, which used to reside in the Edit menu, and remains accessible via key assignment and its macro function.

See also: Move Line Up

4.2.51 Math -> Increment

Menu: Edit > Math > Increment

Default Shortcut Key: none

Macro function: Increment()

The Increment command can be used to increment an integer value (ie, a whole number, not a floating point value) at the text cursor by another integer value. A dialog box will appear to retrieve the value to be added. After clicking 'OK' the arithmetic is performed, and the old value is replaced by the result.

If the text cursor is situated on a character rather than a numeric value, the supplied value will be added to the character at the cursor and the new character will be displayed. If the resultant character value is out of range, an error message will be given.

4.2.52 Math -> Decrement

Menu: Edit > Math > Decrement

Default Shortcut Key: none

Macro function: Decrement()

The Decrement command can be used to decrement an integer value (ie, a whole number, not a floating point value) at the text cursor by another integer value. A dialog box will appear to retrieve the value to be subtracted. After clicking 'OK' the arithmetic is performed, and the old value is replaced by the result.

If the text cursor is situated on a character rather than a numeric value, the supplied value will be subtracted from the character at the cursor and the new character will be displayed. If the resultant character value is out of range, an error message will be given.

4.2.53 Math -> Multiply

Menu: Edit > Math > Multiply

Default Shortcut Key: none

Macro function: Multiply()

The Multiply command can be used to multiply an integer value (ie, a whole number, not a floating point value) at the text cursor by another integer value. A dialog box will appear to retrieve the value to multiply by. After clicking 'OK' the arithmetic is performed, and the old value is replaced by the result.

If the text cursor is situated on a character rather than a numeric value, the character value of the character at the cursor will be multiplied by the supplied value and the resultant character will be displayed. If the resultant character value is out of range, an error message will be given.

4.2.54 Math -> Divide

Menu: Edit > Math > Divide

Default Shortcut Key: none

Macro function: Divide()

The Divide command can be used to divide an integer value (ie, a whole number, not a floating point value) at the text cursor by another integer value. The result will be displayed as an integer value. A dialog box will appear to retrieve the value to divide by. After clicking 'OK' the arithmetic is performed, and the old value is replaced by the result.

If the text cursor is situated on a character rather than a numeric value, the character value of the character at the cursor will be divided by the supplied value and the resultant character will be displayed. If the resultant character value is out of range, an error message will be given.

4.2.55 Swap Words

Menu: Edit > Swap Words

Default Shortcut Key: Shift+F4

Macro function: SwapWords()

The Swap Words command can be used to swap the word at the text cursor with the word to its right.

The characters which serve to delimit words can be set on the Configure | Preferences | Cursor options page. The option is titled *These characters will delimit words*.

The last word on a line cannot be swapped, since the Swap Words command does not span lines.

4.2.56 Swap Lines

Menu: none

Default Shortcut Key: F4

Macro function: SwapLines()

The Swap Lines command exchanges the current line with the line below. The text cursor remains on the same line it was on before the command was issued. This command is useful for swapping the order of a pair of items within an ordered list.

The last line in the file is not eligible for this operation.


This command used to reside in the Edit menu, but has been replaced by the Move Line Up and Move Line Down commands. Though not accessible directly from the menu, the command remains active internally, and can be accessed by a key assignment, and via its macro function.

4.2.57 Flip Case

Menu: Edit > Flip Case

Default Shortcut Key: Shift+Ctrl+F

Macro function: FlipCase()

The Flip Case command flips (toggles, inverts) the case of the character at the text cursor, and moves the cursor to the next character in the line. Use this command to quickly convert a short string of lowercase characters to uppercase, or uppercase characters to lowercase, by issuing the command repeatedly to move through the string.

4.3 Block Menu

4.3.1 Select Stream

Menu: Block > Select Stream

Default Shortcut Key: Alt+1

Macro function: SelectStream()

Stream Selection
The Select Stream command is used to set Boxer's *default selection mode* to Stream. In this mode text is selected by lines, as opposed to rectangular blocks as can be achieved with Select Columnar. Stream selection is the conventional type of text selection found in most text editors and word processors.

```
if (ListView1->Selected != NULL)
    {
    // build a file path to the selected macro file
    SourceFile = MacroDir + ListView1->Selected->Caption + MACRO_EXT;

    isedited = (MainForm->IsAnEditedFile(SourceFile) != NULL);

    // test to prevent unnecessary reloading of file
    if (isedited || (EditorFileName != ListView1->Selected->Caption))
        {
        EditorFileName = ListView1->Selected->Caption;
```

The *default selection mode* determines what selection style is used when text selection is initiated either from the keyboard or with the left mouse button. If the *default selection mode* is Columnar, a temporary override to Stream can be achieved by holding down the *Shift* key before selecting with the mouse.

Selecting Text with the Keyboard
Text selection can be performed by keyboard by pressing and holding down the *Shift* key and moving the text cursor. Any cursor movement key can be used to move the cursor, e.g., PgUp, PgDn, Home, End, etc, so long as the *Shift* key remains depressed. As the text cursor is moved through the file, the selected area is extended.

Selecting Text with the Mouse
Text selection can be accomplished with the mouse by pressing and holding the left mouse button and dragging the mouse to highlight the desired text. As the mouse pointer is moved through the file, the selected area is extended. An existing selection can be extended by depressing *Shift* or *Ctrl* before clicking or dragging the mouse.

Text can also be selected by clicking or dragging the mouse in the area to the left of column one. A single line can be selected by clicking the left mouse button at the far left edge of the line. Multiple lines can be selected by dragging the mouse in this area. An existing selection can be extended by depressing *Shift* before clicking in this area.

A word can be selected by double clicking anywhere within the word.

Selecting the Entire File
The entire file can be selected by issuing the Select All Text command.

Ending Text Selection
Text selection ends when the mouse button or the *Shift* key is released. The selected text remains highlighted and is ready to be operated upon by any of Boxer's Block commands, such as Cut, Copy, Append, Cut-Append, etc.

Extending a Selection with Other Commands
When text is selected, the Go to Line and Find commands provide options to extend the selection to the new cursor position that results from their operation. The Replace command provides an option to limit the replacement operation to the selected text.

Saving and Printing Text Selections

When text is selected, issuing either the Save or Save Selection As command prompts the user to save the selected area to a disk file. Similarly, when text is selected and the Print command is issued, an option is available to print only the selected area.

Deselecting Text
Selected text can be deselected by clicking once with the left mouse button, pressing the Escape key, or pressing any of the cursor arrow keys. Pressing the *Enter* key, or any other alphanumeric key will cause selected text to be deleted.

If a stream selection is started in the virtual space beyond the end of a line, or within the virtual space of a preceding Tab character, the starting column of the selection will be moved to the nearest column to the left which contains a character.

If you start a selection and find that the mode was set to Columnar, simply issue this command to convert the existing selection to Stream. There's no need to cancel a selection; the selection mode can be toggled between Stream and Columnar at will.

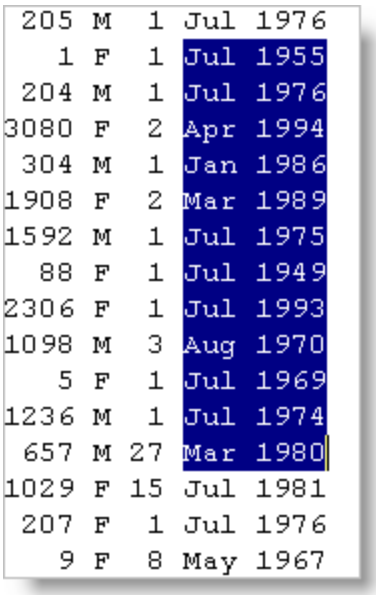
4.3.2 Select Columnar

Menu: Block > Select Columnar

Default Shortcut Key: Alt+2

Macro function: SelectColumnar()

Columnar Selection
The Select Columnar command is used to set Boxer's *default selection mode* to Columnar. In this mode text is selected in rectangular blocks, as opposed to lines.



The *default selection mode* determines what selection style is used when text selection

is initiated either from the keyboard or with the left mouse button. If the *default selection mode* is Stream, a temporary override to Columnar can be achieved by holding down the *Ctrl* key before selecting with the mouse. If a three-button mouse is installed, the center button can be used to perform Columnar text selection regardless of the current selection mode.

Selecting Text with the Keyboard
Text selection can be performed by keyboard by pressing and holding down the *Shift* key and moving the text cursor. Any cursor movement key can be used to move the cursor, e.g., PgUp, PgDn, Home, End, etc., as long as the *Shift* key remains depressed. As the text cursor is moved through the file, the selected area is extended.

Selecting Text with the Mouse
Text selection can be accomplished with the mouse by pressing and holding the left (or center) mouse button and dragging the mouse to highlight the desired text. As the mouse pointer is moved through the file, the selected rectangle is extended. An existing selection can be extended by depressing *Shift* or *Ctrl* before clicking or dragging the mouse.

Selecting the Entire File
The entire file can be selected by issuing the Select All Text command. Doing so while in Columnar mode will result in the default selection mode being changed to Columnar.

Ending Text Selection
Text selection ends when the mouse button or the *Shift* key is released. The selected text remains highlighted and is ready to be operated upon by any of Boxer's Block commands, such as Cut, Copy, Append, Cut-Append, etc.

Extending a Selection with Other Commands
When selected text is present, the Go to Line and Find commands provide options to extend the selection to the new cursor position that results from their operation. The Replace command provides an option to limit the replacement operation to the selected text.

Saving and Printing Text Selections
When text is selected, issuing either the Save or Save Selection As command prompts the user to save the selected area to a disk file. Similarly, when text is selected and the Print command is issued, an option is available to print only the selected area.

Deselecting Text
Selected text can be deselected by clicking once with the left mouse button, pressing the Escape key, or pressing any of the cursor arrow keys. Pressing *Enter*, or any other alphanumeric key will cause selected text to be deleted.


If a columnar selection is started in the virtual space beyond the end of a line, or within the virtual space of a preceding Tab character, the starting column of the selection will be moved to the nearest column to the left which contains a character.


If you start a selection and find that the mode was set to Stream, simply issue this command to convert the existing selection to Columnar. There's no need to cancel a selection; the selection mode can be toggled between Stream and Columnar at will.

4.3.3 Select without Shift

Menu: Block > Select without Shift

Default Shortcut Key: Alt+M

Macro function: SelectWithoutShift()

This command can be used to initiate a text selection mode in which the *Shift* key does not need to be kept depressed during selection, as is the custom under Windows. After this command is issued, any cursor movement command can be used to extend the selection as desired. The text selection can be released by pressing *Escape*.

The type of selection that results is determined by the current selection mode, which can be set using the Select Columnar and Select Stream commands.


Former users of Boxer/DOS may welcome this command as it provides a method of text selection that approximates the operation of our earlier products.

4.3.4 Indent One Space

Menu: Block > Indent One Space

Default Shortcut Key: Ctrl+Space

Macro function: IndentOneSpace()

The Indent One Space command causes a selected range of lines to be indented by one space. On lines which already contain one or more Tab characters of indent, the space character will be applied to the right of the existing indent so that the expected effect is achieved.

If no lines are selected, indentation will be performed on the current line only.


Regardless of the shortcut key assigned to this command, the *Space* key will always perform a block indent when a range of lines is selected. If a small selection is present on a single line the selection will be replaced with a Space character.

4.3.5 Indent One Tabstop

Menu: Block > Indent One Tabstop

Default Shortcut Key: Shift+Tab

Macro function: IndentOneTabstop()

The Indent one Tabstop command causes a selected range of lines to be indented by one tabstop. Tab options may be set using the Configure | Preferences | Tabs options page.

If no lines are selected, indentation is performed on the current line only.


Pressing the key assigned to the Insert Tab command will also serve to indent a selected range of lines. In most of the keyboard layouts which accompany Boxer the *Tab* key is assigned to the Insert Tab command.

4.3.6 Indent with String

Menu: Block > Indent with String

Default Shortcut Key: none

Macro function: IndentWithString()

The Indent with String command can be used to simultaneously indent a range of selected lines, and fill the indent region with a user-specified text string.

4.3.7 Unindent

Menu: Block > Unindent

Default Shortcut Key: Shift+Backspace

Macro function: Unindent()

The Unindent command causes a selected range of lines to be unindented by one space or one tab character. If the range of lines to be unindented contains lines with varying levels of indent--or mixed indents of spaces and tabs--the Unindent command can still be used without concern. Once all of the *whitespace* on a given line has been deleted, Unindent will not remove additional characters, though it will continue to operate on other lines within the selection with remaining indent.

If text is not selected, Unindent will act upon the current line.

4.3.8 Convert Case -> Upper

Menu: Block > Convert Case > Upper

Default Shortcut Key: none

Macro function: CaseUpper()

The Upper command converts alphabetic characters within the selected text to uppercase.

```
Before conversion: The quick brown fox jumped over the lazy dog.
```

```
After conversion : THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG.
```

Boxer consults the case conversion map provided by the operating system so that accented characters can be properly converted.

4.3.9 Convert Case -> Lower

Menu: Block > Convert Case > Lower

Default Shortcut Key: none

Macro function: CaseLower()

The Lower command converts alphabetic characters within the selected text to lowercase.

```
Before conversion: The quick brown fox jumped over the lazy dog.
After conversion : the quick brown fox jumped over the lazy dog.
```

Boxer consults the case conversion map provided by the operating system so that accented characters can be properly converted.

4.3.10 Convert Case -> Invert

Menu: Block > Convert Case > Invert

Default Shortcut Key: none

Macro function: CaseInvert()

The Invert command converts alphabetic characters within the selected text to the opposite case. Uppercase characters are converted to lowercase; lowercase characters are converted to uppercase.

```
Before conversion: The quick brown fox jumped over the lazy dog.
After conversion : tHE QUICK BROWN FOX JUMPED OVER THE LAZY DOG.
```

Boxer consults the case conversion map provided by the operating system so that accented characters can be properly converted.

4.3.11 Convert Case -> Words

Menu: Block > Convert Case > Words

Default Shortcut Key: none

Macro function: CaseWords()

The Words command converts the first character of all words within the selected text to uppercase.

```
Before conversion: The quick brown fox jumped over the lazy dog.
After conversion : The Quick Brown Fox Jumped Over The Lazy Dog.
```

Boxer consults the case conversion map provided by the operating system so that accented characters can be properly converted.

☞ By default, this command will first convert the selected text to lowercase before capitalizing each word. This ensures that the command operates as expected when processing text in all uppercase. Converting to lowercase may disrupt some all-caps words (such as acronyms) that should have remained in uppercase, so you should review the results for accuracy after applying the conversion. If you prefer that the selected text not be forced to lowercase prior to operation, you can change this behavior on the Configure | Preferences | Editing 2 dialog page.

4.3.12 Convert Case -> Sentences

Menu: Block > Convert Case > Sentences

Default Shortcut Key: none

Macro function: CaseSentences()

The Sentences command converts the first character of all sentences within the selected text to uppercase. For purposes of this command, a sentence is considered to be a series of words that ends with either a period, a question mark, or an exclamation mark.

```
Before conversion: the quick brown fox jumped over the lazy dog.
After conversion : The quick brown fox jumped over the lazy dog.
```

Boxer consults the case conversion map provided by the operating system so that accented characters can be properly converted.

☞ By default, this command will first convert the selected text to lowercase before capitalizing each sentence. This ensures that the command operates as expected when processing text in all uppercase. Converting to lowercase may disrupt some all-caps words (such as acronyms and proper nouns) that should have remained in uppercase, so you should review the results for accuracy after applying the conversion. If you prefer that the selected text not be forced to lowercase prior to operation, you can change this behavior on the Configure | Preferences | Editing 2 dialog page.

4.3.13 Convert Case -> Title

Menu: Block > Convert Case > Title

Default Shortcut Key: none

Macro function: CaseTitle()

This command converts the selected text to conform to the rules of title case (aka proper case). Grammar experts do not all agree on the precise rules for title case, but most references use these rules:

1. Always capitalize the first word
2. Always capitalize the last word
3. Capitalize all other words, except articles, prepositions and conjunctions which have fewer than five letters

☞ Because the application of title case presumes a knowledge of the language--capitalization depends on parts of speech--this command is limited to operating on English text.

☞ By default, this command will first convert the selected text to lowercase before capitalizing words. This ensures that the command operates as expected when processing text in all uppercase. Converting to lowercase may disrupt some all-caps words (such as acronyms) that should have remained in uppercase, so you should review the results for accuracy after applying the conversion. If you prefer that the selected text not be forced to lowercase prior to operation, you can change this behavior on the Configure | Preferences | Editing 2 dialog page.

4.3.14 Convert Other -> Tabs to Spaces

Menu: Block > Convert Other > Tabs to Spaces

Default Shortcut Key: none

Macro function: TabsToSpaces()

The Tabs to Spaces command can be used to convert the Tabs within a selected area of text into an equivalent number of Spaces. In doing so, Boxer uses the current display value of a Tab (View | Tab Display Size) to determine how many Spaces should be used.

☞ No attempt is made to determine whether the Tabs being changed reside within a quoted string. Programmers should be careful when using this command, as changing Tabs within a quoted string my yield undesirable results if the string was to be displayed in a message on-screen.

☞ The Spaces to Tabs and Tabs to Spaces commands are not opposites. If the Tab Display Size is 4, the sentence:

```
The<tab>dog<tab>ran<tab>wild.
```

would be converted by the Tabs to Spaces command to:

```
The<space>dog<space>ran<space>wild.
```

Running the Spaces to Tabs command on this sentence would not yield the original sentence. A sequence of two or more spaces is required before a tab character is considered for placement into the converted text.


If the Tab Display Size command has been used to designate an additional tab character for display purposes, the Tabs to Spaces command will treat that character as a tab when performing its conversion to spaces. This makes it possible to convert a data file that uses a character-separated field format into a fixed width field format. See Converting CSV Data to Fixed Width for more details.

4.3.15 Convert Other -> Spaces to Tabs

Menu: Block > Convert Other > Spaces to Tabs

Default Shortcut Key: none

Macro function: SpacesToTabs()

The Spaces to Tabs command can be used to convert the Spaces within a selected area of text into an equivalent number of Tabs. In doing so, Boxer uses the current display value of a Tab (View | Tab Display Size) to determine how many Tabs should be used.


No attempt is made to determine whether the Spaces being changed reside within a quoted string. Programmers should be careful when using this command, as changing Spaces within a quoted string may yield undesirable results if the string was to be displayed in a message on-screen.


The Spaces to Tabs and Tabs to Spaces commands are not opposites. If the Tab Display Size is 4, the sentence:

```
The<tab>dog<tab>ran<tab>wild.
```

would be converted by the Tabs to Spaces command to:

```
The<space>dog<space>ran<space>wild.
```

Running the Spaces to Tabs command on this sentence would not yield the original sentence. A sequence of two or more spaces is required before a tab character is considered for placement into the converted text.

4.3.16 Convert Other -> OEM to ANSI

Menu: Block > Convert Other > OEM to ANSI

Default Shortcut Key: none

Macro function: OEMtoANSI()

The OEM to ANSI command converts characters within the selected text from OEM (ASCII) character encoding to ANSI character encoding. These character encoding schemes share all the common alphabetic and numeric character mappings, but differ in the area of accented and/or graphic characters. A conversion may be appropriate when a file which was created with a DOS program must be prepared for use with a Windows program. Note that not all characters will have equivalents in the destination character set. In such cases, a conversion will not be made for that character.

Boxer's OEM Chart and ANSI Chart commands can be useful for viewing the character assignments in each of these encoding schemes.

4.3.17 Convert Other -> ANSI to OEM

Menu: Block > Convert Other > ANSI to OEM

Default Shortcut Key: none

Macro function: ANSItoOEM()

The ANSI to OEM command converts characters within the selected text from ANSI character encoding to OEM (ASCII) character encoding. These character encoding schemes share all the common alphabetic and numeric character mappings, but differ in the area of accented and/or graphic characters. A conversion may be appropriate when a file which was created with a Windows program must be prepared for use with a DOS program. Note that not all characters will have equivalents in the destination character set. In such cases a conversion will not be made for that character.

Boxer's ANSI Chart and OEM Chart commands can be useful for viewing the character assignments in each of these encoding schemes.

4.3.18 Convert Other -> EBCDIC to ASCII

Menu: Block > Convert Other > EBCDIC to ASCII

Default Shortcut Key: none

Macro function: EBCDICtoASCII()

This command will convert text encoded in the EBCDIC character set to the ASCII character set. The text to be converted must first be selected. If an entire file is to be converted, use the Select All Text command to select the whole file.

EBCDIC is a character encoding system used primarily on mainframe computers. The ASCII character encoding system is used widely on personal computers. At times, a file that uses EBCDIC encoding may need to be converted for use on a computer that uses the ASCII character encoding system. This command can be used for that purpose.

 EBCDIC is an acronym for Extended Binary Coded Decimal Interchange Code.

☞ ASCII is an acronym for American Standard Code for Information Interchange.

4.3.19 Convert Other -> ASCII to EBCDIC

Menu: Block > Convert Other > ASCII to EBCDIC

Default Shortcut Key: none

Macro function: ASCIItoEBCDIC()

This command will convert text encoded in the ASCII character set to the EBCDIC character set. The text to be converted must first be selected. If an entire file is to be converted, use the Select All Text command to select the whole file.

EBCDIC is a character encoding system used primarily on mainframe computers. The ASCII character encoding system is used widely on personal computers. At times, a file that uses ASCII encoding may need to be converted for use on a computer that uses the EBCDIC character encoding system. This command can be used for that purpose.

☞ ASCII is an acronym for American Standard Code for Information Interchange.

☞ EBCDIC is an acronym for Extended Binary Coded Decimal Interchange Code.

4.3.20 Convert Other -> ROT5

Menu: Block > Convert Other > ROT5

Default Shortcut Key: none

Macro function: ROT5()

This command will apply a ROT5 (rotation 5) conversion to the selected text. If an entire file is to be converted, use the Select All Text command to select the whole file.

ROT5 is a simple substitution cypher that replaces each digit (0-9) with the digit that resides 5 positions higher in sequence. The digit '0' would be replaced by '5', '1' by '6', and so on. When adding 5 would exceed the digit '9', the conversion wraps back to the beginning of the sequence.

ROT5 is sometimes used to disguise numeric text from casual viewing, such as when the answer to a puzzle is presented alongside the question. Applying the conversion to ROT5-encoded text a second time returns the original text. Additional information about ROT13 and other cyphers can be found on Wikipedia.

4.3.21 Convert Other -> ROT13

Menu: Block > Convert Other > ROT13

Default Shortcut Key: none

Macro function: ROT13()

This command will apply a ROT13 (rotation 13) conversion to the selected text. If an entire file is to be converted, use the Select All Text command to select the whole file.

ROT13 is a simple text substitution cypher that replaces each alphabetic character (A-Z and a-z) with the character that resides 13 positions forward in the alphabet. The letter 'A' would be replaced by 'N', 'B' by 'O', and so on. When adding 13 would exceed the letter 'Z', the conversion wraps back to the beginning of the alphabet.

ROT13 is sometimes used to disguise text from casual viewing, such as when the answer to a puzzle is presented alongside the question. Some entries in the *Windows Registry* are ROT13 encoded. Applying the conversion to ROT13-encoded text a second time returns the original, human-readable text. Additional information about ROT13 can be found on Wikipedia.

4.3.22 Convert Other -> ROT18

Menu: Block > Convert Other > ROT18

Default Shortcut Key: none

Macro function: ROT18()

This command will apply a ROT18 (rotation 18) conversion to the selected text. If an entire file is to be converted, use the Select All Text command to select the whole file.

ROT18 is a simple text substitution cypher that replaces each alphabetic character (A-Z and a-z) with the character that resides 13 positions forward in the alphabet, and each digit (0-9) with the digit that resides 5 positions higher in sequence. A ROT18 conversion is thus equivalent to applying the ROT5 and ROT13 conversion commands to the same text.

ROT18 is sometimes used to disguise text from casual viewing, such as when the answer to a puzzle is presented alongside the question. Applying the conversion to ROT18-encoded text a second time returns the original, human-readable text. Additional information about ROT13 and other cyphers can be found on Wikipedia.

4.3.23 Convert Other -> ROT47

Menu: Block > Convert Other > ROT47

Default Shortcut Key: none

Macro function: ROT47()

This command will apply a ROT47 (rotation 47) conversion to the selected text. If an entire file is to be converted, use the Select All Text command to select the whole file.

ROT47 is a simple text substitution cypher that replaces each character value in the ASCII range 33 to 126 inclusive with the character whose value is 47 positions higher.

When adding 47 would exceed character value 126, the conversion wraps back to the beginning of the sequence.

ROT47 is sometimes used to disguise text from casual viewing, such as when the answer to a puzzle is presented alongside the question. Applying the conversion to ROT47-encoded text a second time returns the original, human-readable text. Additional information about ROT13 and other cyphers can be found on Wikipedia.

4.3.24 Comment

Menu: Block > Comment

Default Shortcut Key: F5

Macro function: Comment()

The Comment command can be used to apply commenting to the current line--or to a selected range of lines--when editing a file for which Boxer has syntax information defined (see Configure | Syntax Highlighting).

When text is not selected, the current line will be commented using the end-of-line comment sequence for the language being edited. If that sequence is not available, the entire line will be enclosed using the open and close comment sequences. In either case the text cursor is advanced to the line below following the operation.

If text is selected, the selected lines will be bracketed with the open and close block comment sequences for the language being edited. If the language does not support block commenting, the end-of-line comment sequence will be applied to each line within the selected range. If neither of these sequences has been defined, an error message will be given.

The Uncomment command can be used to remove commenting from the current line or from selected text.

4.3.25 Uncomment

Menu: Block > Uncomment

Default Shortcut Key: Shift+F5

Macro function: Uncomment()

The Uncomment command will remove commenting from the current line--or the selected text--according to Boxer's syntax information about the language being edited (see Configure | Syntax Highlighting).

The Comment command can be used to apply commenting to the current line or to selected text.

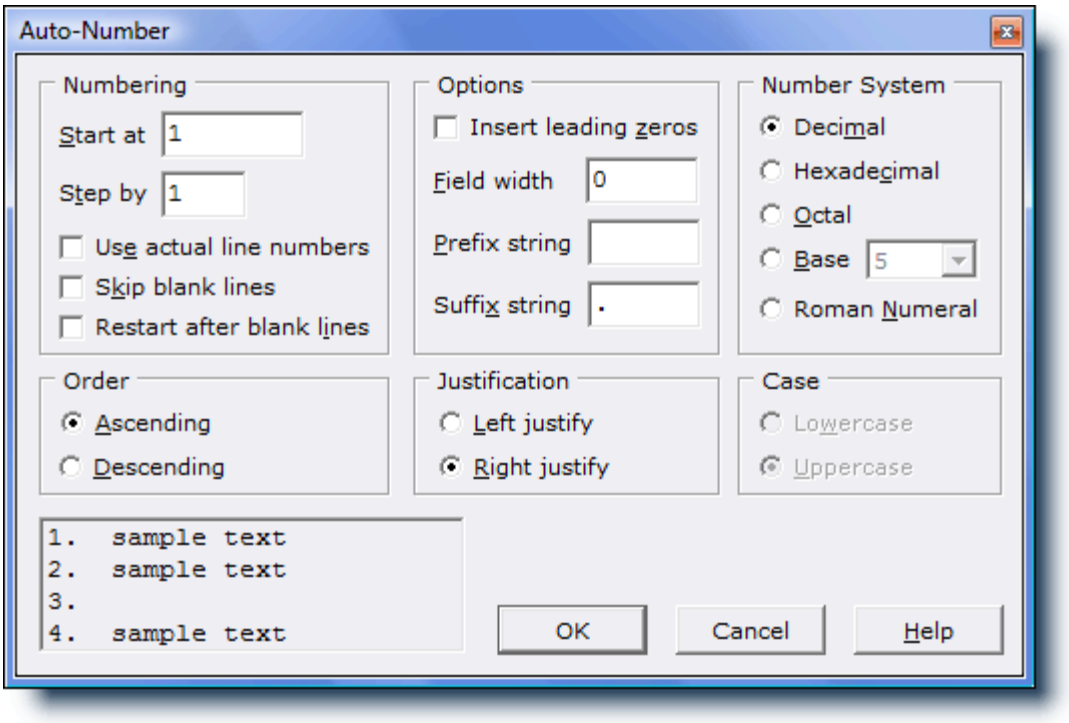
4.3.26 Auto-Number

Menu: Block > Auto-Number

Default Shortcut Key: None

Macro function: AutoNumber()

The Auto-Number command can be used to automatically number a selected range of lines. A variety of options are available to control the numbering operation, and are described below:



Numbering

Start at
This is the value that will be used to start the numbering from.

Step by
This is the increment that numbering will jump by from line to line. Programmers might use 10 or 100 for program listings, for example.

Use actual line numbers
This option allows the line number for each line to be used. When this option is used, the *Start at* and *Step by* options are disabled.

Skip blank lines
Use this option to control whether or not blank lines will be numbered.

Restart after blank lines
This option causes line numbering to restart from the starting value after a sequence of one or more blank lines is encountered.

Order

Ascending
Use this option for numbering which increases in value.

Descending
Use this option for numbering which decreases in value.

Options

Insert leading zeros
This option can be used to force leading zeros on the numbers.

Field width
Use the field width property to control the width of the numbers that will be generated. Use '0' for automatic sizing.

Prefix string
Use this edit box to specify the text to be placed at the left of the numbers.

Suffix string
Use this edit box to specify the text to be placed at the right of the numbers.

Justification

Left justify
This option can be used for numbering which is left aligned.

Right justify
This option can be used for numbering which is right aligned.

Number System

Decimal
Use this option for numbering in the *decimal* system.

Hexadecimal
Use this option for numbering in the *hexadecimal* system. The case of the alphabetic characters used can be controlled with the *Lowercase* and *Uppercase* options below.

Octal
Use this option for numbering in the *octal* system.

Base
Use this combobox to select any base in the range 2 to 36. For bases 11 and above, alphabetic characters are used in place of digits. The case of the alphabetic characters used can be controlled with the *Lowercase* and *Uppercase* options below.

Roman Numeral
Use this option for numbering with Roman Numerals. The case of the alphabetic characters used can be controlled with the *Lowercase* and *Uppercase* options below.

Case

Lowercase
This option can be used to dictate that lowercase characters be used when *Hexadecimal* or *Roman Numeral* numbering is in use.

Uppercase
This option can be used to dictate that uppercase characters be used when *Hexadecimal* or *Roman Numeral* numbering is in use.

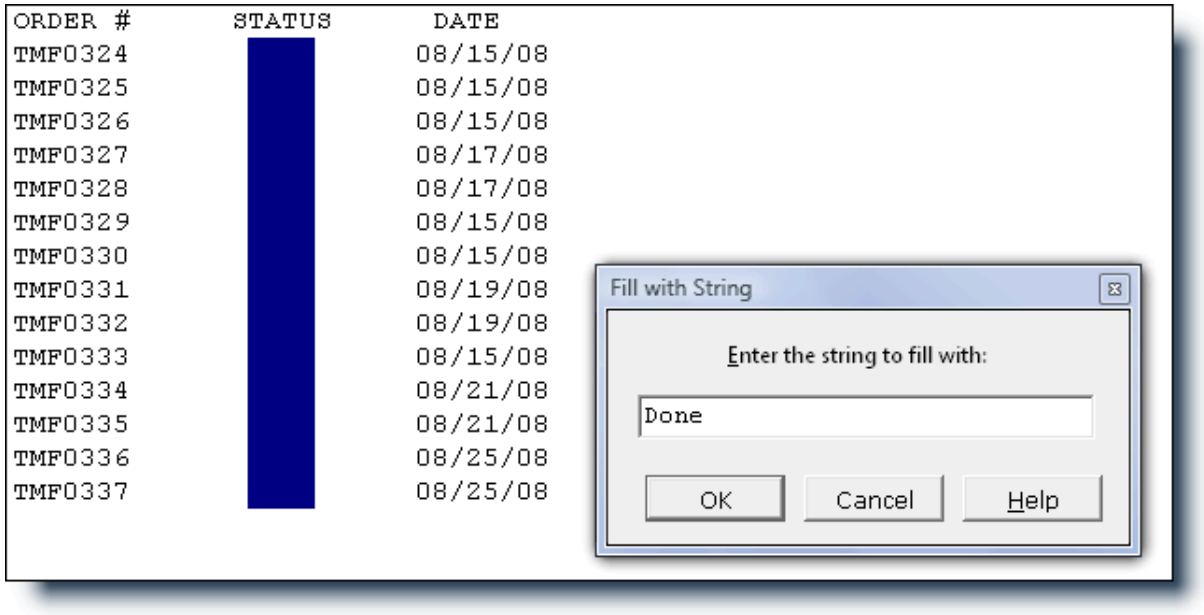
4.3.27 Fill with String

Menu: Block > Fill with String

Default Shortcut Key: none

Macro function: FillWithString()

The Fill with String command can be used to fill a selected area with a supplied text string. The selected area will be filled automatically with the string supplied. If the text supplied is less than the width of the selected text, the pattern will be repeated to fill the selected area.



To prevent unexpected results, the lines affected are automatically de-tabbed prior to performing the Fill operation. If any lines in the selected range are too short, they will

be extended so that a complete fill of the selected area is achieved.


Special characters can be entered into the Fill with String edit box using the technique described in the Help topic Inserting Special Characters.

4.3.28 Invert Lines

Menu: Block > Invert Lines

Default Shortcut Key: none

Macro function: InvertLines()

The Invert Lines command can be used to invert a range of selected lines. The last selected line will become first, and the first selected line will become the last.

For example, the text:

```
 1. Colorado Springs, CO
 2. Denver, CO
 3. Eugene, OR
 4. Las Vegas, NV
 5. Los Angeles, CA
 6. Oakland, CA
 7. Phoenix, AZ
 8. Portland, OR
 9. Pueblo, CO
10. Riverside, CA
```

would become:

```
10. Riverside, CA
 9. Pueblo, CO
 8. Portland, OR
 7. Phoenix, AZ
 6. Oakland, CA
 5. Los Angeles, CA
 4. Las Vegas, NV
 3. Eugene, OR
 2. Denver, CO
 1. Colorado Springs, CO
```

4.3.29 Line Spacing

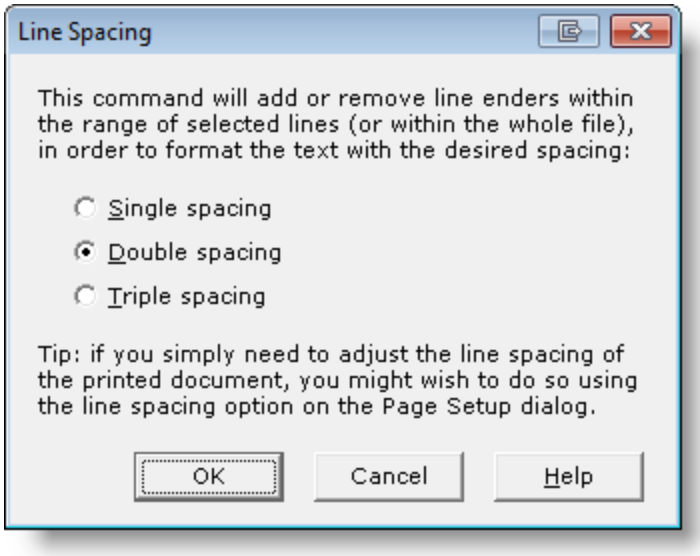
Menu: Block > Line Spacing

Default Shortcut Key: none

Macro function: LineSpacing()

The Line Spacing command allows a range of selected lines to be converted to single-,

double- or triple-spaced format. If no selection is present, the operation will be performed across the entire file.



If you simply need to adjust the line spacing of the printed document, you might wish to do so using the line spacing option on the Page Setup dialog.

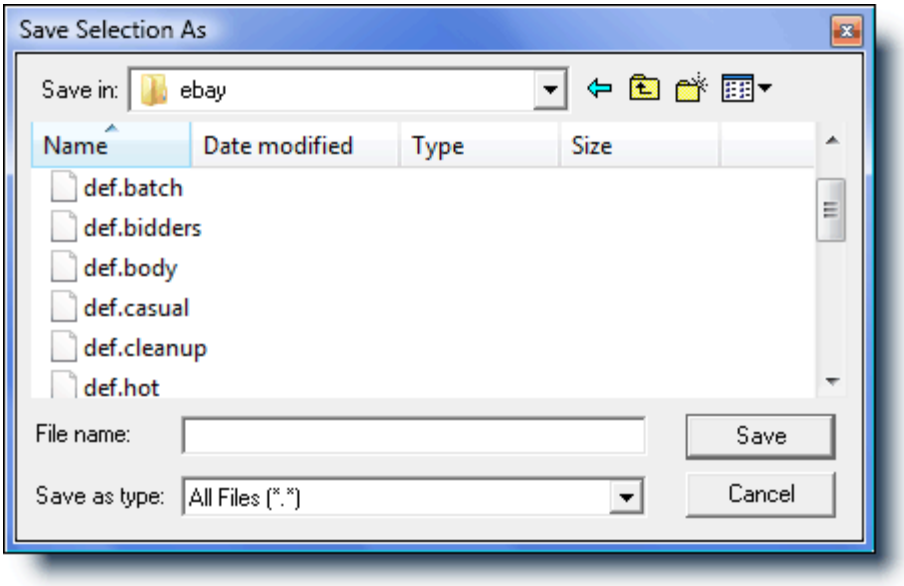
4.3.30 Save Selection As

Menu: Block > Save Selection As

Default Shortcut Key: none

Macro function: SaveSelectionAs()

The Save Selection As command brings up the standard Windows Save dialog and prompts the user for a filename under which to save the selected text.



- Boxer will not automatically add a file extension to the filename you provide; you should add the desired file extension yourself.

- The current file encoding and line ender style, as indicated in File Properties, will be used for the file that's created. If you want to save the selected text to a file using different encoding or line enders, paste the text into a new file and set the properties of the new file before saving.

- If the Save command is issued while text is selected, a dialog box can appear to get the name of the file to which the selected text should be saved. This option is off by default, but can be enabled on the Configure | Preferences | File I/O options page. The option is titled *File Save performs Save Selection As when text is selected*. This option page also contains other configuration options which relate to loading and saving files.

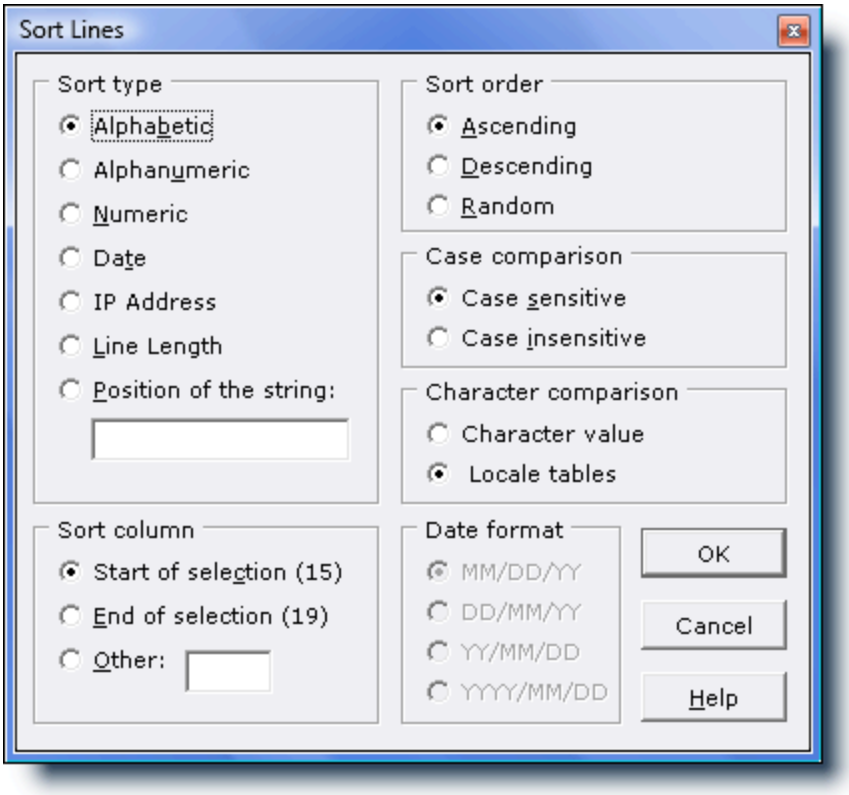
4.3.31 Sort Lines

Menu: Block > Sort Lines

Default Shortcut Key: none

Macro function: SortLines()

The Sort Lines command can be used to sort a range of selected lines. A variety of options are provided to control the nature of the sort performed. These options are described below:



Sort Type

Alphabetic
Use this option to sort alphabetically. Any digits that appear within the data will not be treated as numeric values.

Alphanumeric
This option sorts alphabetically, but embedded sequences of digits within the data are treated as numeric values. This is sometimes called a *natural* sort, and is best understood with an example. Consider the results of the Alphabetic and Alphanumeric sorting options when applied to a list of filenames:

Alphabetic sort:	**Alphanumeric sort:**
z1.txt	z1.txt
z10.txt	z2.txt
z100.txt	z3.txt
z101.txt	z4.txt
z102.txt	z5.txt
z11.txt	z6.txt
z12.txt	z7.txt
z19.txt	z8.txt
z2.txt	z9.txt
z20.txt	z10.txt
z3.txt	z11.txt
z4.txt	z12.txt
z5.txt	z19.txt

```
z6.txt                  z200.txt
z7.txt                  z100.txt
z8.txt                  z101.txt
z9.txt                  z102.txt
```

In the left column, the results of an Alphabetic sort are shown. On the right, the more pleasing results of an Alphanumeric are displayed.

Numeric
Use this option to sort numerically. The data will be interpreted as numbers, and not as character data. Alphabetic data will sort as though its value were zero.

Date
Use this option to sort chronologically by date. Be sure to set the proper Date Format in the options box provided.

IP Address
Use this option to sort IP Addresses data with proper consideration to each node within the address.

Line Length
Use this option to sort lines according to their length.

Position of the string:
This option allows data to be sorted based on the position of a supplied string within the data. This option can be useful for segregating lines of data that contain a certain type of information. For example, by sorting on the string '@', lines containing email addresses would be isolated from those lines not containing email addresses.

Sort Column

Start of selection
Use this option if the sort should be performed based on the starting column of the selection. The column number of the start of the selection is shown in parentheses to the right.

End of selection
Use this option if the sort should be performed based on the ending column of the selection. The column number of the end of the selection is shown in parentheses to the right.

Other
Use this option if the sort should be performed based on some other column in the data.

Sort Order

Ascending
Use this option to sort in increasing order.


The sort command will consult the current locale so that accented characters are

sorted according to the local collating sequence.

Descending
Use this option to sort in decreasing order.


The sort command will consult the current locale so that accented characters are sorted according to the local collating sequence.

Random
Use this option to sort randomly. This option might be used to randomly order a list which was already sorted. When this option is selected, all other options on the dialog become irrelevant.

Case Comparison

Case Sensitive
When an alphabetic or alphanumeric sort is being performed, this option can be used to ensure that character case is considered significant.

Case Insensitive
When an alphabetic or alphanumeric sort is being performed, this option can be used to ensure that character case is ignored.

Character Comparison

Character Value
When an alphabetic or alphanumeric sort is being performed, this option causes characters to be compared based on their actual character values. This is sometimes called an 'ASCII sort.'

Locale Tables
When an alphabetic or alphanumeric sort is being performed, this option causes characters to be compared using the 'locale tables' supplied by the operating system. Using the locale tables ensures that when accented characters are encountered in the data being sorted, they will be sorted according to local custom.

The results of a Locale Table sort can vary from those achieved using Character Value. A case sensitive sort using Character Value would yield the following result:

```
AAA
BBB
CCC
aaa
bbb
ccc
```

If the same data is sorted with case sensitive and Locale Tables, the following result is achieved:

```
aaa
AAA
bbb
```

BBB
ccc
CCC

☞ A Locale Table sort uses a "word sort," rather than a "string sort." A word sort treats hyphens and apostrophes differently than it treats other symbols that are not alphanumeric, in order to ensure that words such as "coop" and "co-op" stay together within a sorted list.

Date Format

The date options below are applicable when a Date sort is being performed. The slash character is shown for illustration only; any separator symbol--or none at all--may appear in the data being sorted.

MM / DD / YY
Data is formatted with 2-digit month, date and year. This option can also be used for MM / DD / YYYY format dates.

DD / MM / YY
Data is formatted with 2-digit date, month, and year. This option can also be used for DD / MM / YYYY format dates.

YY / MM / DD
Data is formatted with 2-digit year, month, and date.

YYYY / MM / DD
Data is formatted with 4-digit year, 2-digit month and 2-digit date.

4.3.32 Strip HTML/XML Tags

Menu: Block > Strip HTML/XML Tags

Default Shortcut Key: none

Macro function: StripHTMLTags()

The Strip HTML/XML Tags command can be used to remove HTML or XML tags from selected text. HTML tags are markup sequences which appear within the '<' and '>' characters. Boxer does not require that the tag names found within these brackets be legitimate HTML tags. It merely removes any text found to be within such delimiters. In this way, Boxer will be able to process new tags properly as the HTML standard evolves.

☞ **Caution**: If the text being processed contains unbalanced angle bracket characters--specifically an unmated open angle bracket--then all text following the open angle bracket will be treated as an HTML tag, and will be removed.

In addition to stripping HTML tags, the following HTML sequences will be converted to their character equivalents:

```
       <space>
&amp;       &
&quot;      "
&lt;        <
&gt;        >
&ndash;     -
&mdash;     --
&lsquo;     '
&rsquo;     '
&ldquo;     "
&rdquo;     "
&hellip;    ...
```

The conversion of other such sequences is complicated by the fact that accented characters do not map to unique character codes in the ANSI and OEM characters set. These translations are therefore not performed.

4.3.33 Strip Leading Spaces

Menu: Block > Strip Leading Spaces

Default Shortcut Key: none

Macro function: StripLeadingSpaces()

The Strip Leading Spaces command can be used to remove leading Spaces and/or Tabs from the start of each line within the selection.

4.3.34 Strip Trailing Spaces

Menu: Block > Strip Trailing Spaces

Default Shortcut Key: None

Macro function: StripTrailingSpaces()

The Strip Trailing Spaces command can be used to remove trailing Spaces and/or Tabs from the end of each line within the selection. The number of characters removed is reported upon completion.

Trailing blanks can also be stripped automatically when **loading** a file. See the *Strip trailing blanks when loading a file* option on the Configure | Preference | File I/Ooptions page.

Trailing blanks can also be stripped automatically when **saving** a file. See the *Strip trailing blanks when saving a file* option on the Configure | Preference | File I/O options page.

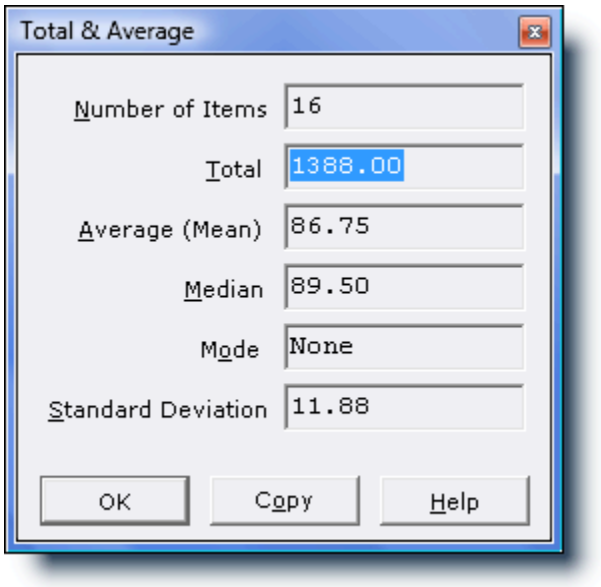
4.3.35 Total and Average

Menu: Block > Total and Average

Default Shortcut Key: None

Macro function: TotalAndAverage()

The Total and Average command can be used to obtain a report on the numeric data contained on a range of selected lines. In addition to computing the total and average (*mean*), the *number of items*, *median*, *mode* and *standard deviation* are also reported.



The *median* is the midpoint between the low and high values in the data.

The *mode* is the most frequently occurring value among the data.

The *standard deviation* provides a measure of the dispersion of the values within the data.

The Total and Average command can be used on ranges of numbers that include thousands separators (commas or periods, typically). The data is analyzed to determine what convention for the thousands and decimal separator is in use. The format of the report is adjusted to reflect the format in use.

Total and Average can be used on values that contain a leading currency symbol. The following symbols will be ignored during computation: dollar sign, yen sign, pound sterling sign, euro sign.

The Total and Average command reports its results using read-only edit boxes so that any of the fields can be copied to the Windows clipboard. The *Copy* button can

be used to copy the full report to the current clipboard.

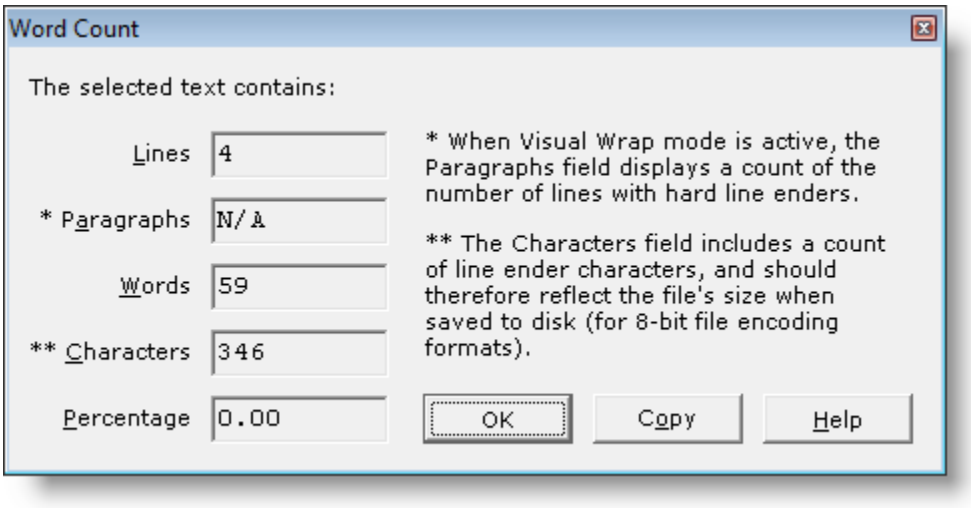
4.3.36 Word Count

Menu: Block > Word Count

Default Shortcut Key: None

Macro function: WordCount()

The Word Count command reports the number of lines, words and characters in the current file, or within the currently selected text. The percentage of the processed text, with respect to the whole file, is also reported.



The Word Count command reports its results using read-only edit boxes so that any of the fields can be copied to the Windows clipboard. The *Copy* button can be used to copy the fill report to the current clipboard.

4.4 Search Menu

4.4.1 Find

Menu: Search > Find

Default Shortcut Key: Ctrl+F

Macro function: Find()

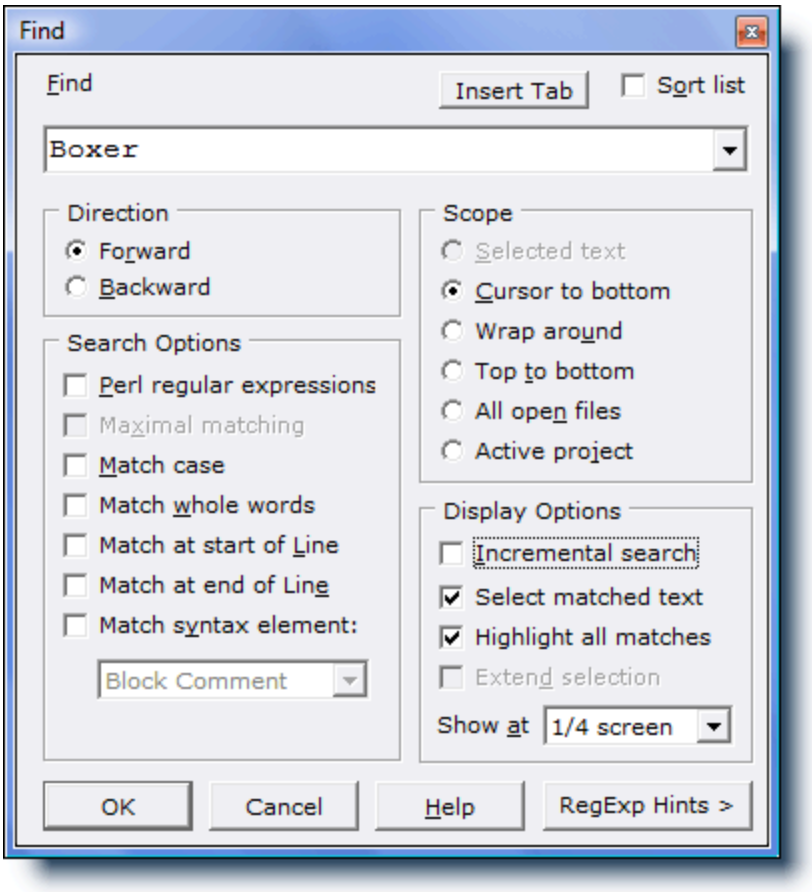
The Find command is used to specify and initiate a search for a text string. Many different options are available to make searching more flexible and more powerful. Wildcard characters (also known as Regular Expressions) can also be used within the

search string.

If the search string is found the text cursor will be moved to the matching string and the text will be selected, if the *Select matched text* option is active. The matched text can then be operated upon as can any other selected text.

If the search string is not found a dialog box will appear to report this fact. If you prefer that this report appear on the message line instead, use the option provided on the Configure | Preferences | Messages options page. The option is titled *Report failed searches in a popup message box*.

The controls and options in the Find dialog box are described below:



Find
This is the edit box where the search string is entered. When the Find command is issued, the word beneath the text cursor is placed into the *Find* edit box, in case that word--or a word which is nearly the same--is to be the search string. The Find Fast command can also be used to search for the word at the text cursor without raising the Find Text dialog. To recall a search string which was previously entered, use the drop-down list or press the up or down arrow keys to review the items in the history list. Regular Expressions may be used within the search string.

The *Delete* key can be used while the drop-down list is displayed to delete a

selected entry from the history list.

Special characters can be entered into the *Find* edit box using the technique described in the Help topic Inserting Special Characters.

Insert Tab
Use this button to insert a tab character into the *Find* edit box.

Ordinarily, the *Tab* key is used to move from field to field within a dialog box. If you would prefer that the Tab key insert a tab character in this dialog box, and in other Find/Replace related dialog boxes, check the relevant box on the Configure | Preferences | Tabs dialog page.

In most fonts, the tab character does not have a unique visual representation. It will often be depicted as an open square box (□), as will be other characters in the low-ASCII portion of the character set.

Sort list
If this box is checked the history list will be maintained in alphabetic order, rather than in the order the strings were entered.

When switching to an alphabetically sorted list, the chronological ordering of the list will be lost, and cannot be restored by unchecking the checkbox.

No attempt is made to associate the history list entries with the time that they were added to the list. If a sorted history list is used consistently, over time the list will come to hold an unrepresentative set of search phrases. In the extreme case, after many Find operations, a list could result that contained only phrases beginning with the letter 'A'. This occurs because entries at the bottom of the list will be removed after the maximum size of the list is reached.

Direction

Forward
This option causes the search to be performed downward, toward the end of file.

Backward
This option causes the search to be performed upward, toward the start of file.

Search Options

Perl regular expressions
If this box is checked, wildcard characters within the search string will be interpreted according to the Perl-Compatible Regular Expression (PCRE) convention. In part, this means that the asterisk (*) will cause a match of zero or more occurrences of the preceding character. The period (.) will match any single character. For more information, see Regular Expressions.

Maximal matching
When using pattern matching characters, there can sometimes be more than one text string that matches the search string. This option can be used to request that the

longest possible matching string be returned.

Match case
This option can be used to force the search string to be matched exactly. When unchecked, a case insensitive search is performed.

Match whole words
This option can be used to restrict matches to those strings which appear as a whole word. The characters which serve to delimit words are user-configurable; see Configure | Preferences | Cursor.

☞ The Match Whole Words option is logically incompatible with the Incremental Search option, and will therefore be disabled when Incremental Search is active.

Match at start of Line
This option can be used to force the search string to be matched only when a matching string appears at the start of a line. This effect can also be achieved with a Regular Expression.

Match at end of Line
This option can be used to force the search string to be matched only when a matching string appears at the end of a line. This effect can also be achieved with a Regular Expression.

Match syntax element
This option can be used to force a matching string to belong to a specified syntax group. Example: you could type the search string 'while' and require that it be matched only when it appears as a Reserved Word. Instances that occur within program comments or quoted strings (or any other syntax) would not be matched. (This is a powerful capability with lots of potential, and one we've never seen in another editor.)

Scope

Selected Text
This option can be used to restrict the search to the extent of the selected text.

Cursor to bottom / Cursor to top
This option causes the search to be performed from the text cursor onward, according to the current direction. The search ends when either the top or bottom of the file is reached.

Wrap around
This option causes the search to be performed from the cursor onward, according to the current direction. When either the top or bottom of file is reached, the search wraps around, and continues to the original cursor position.

☞ When Find Next or Find Previous are used in wrap around mode, a message appears on the status bar when the search has wrapped back to the location of the first match. An option on the Configure | Preferences | Messages dialog page can be used if you prefer that this event be reported in a pop-up message box instead.

Top to bottom / Bottom to top
This option causes the search to be performed from the top or bottom of file onward, according to the current direction.

All open files
This option causes the search to be performed across all open files.

Active project
Use this option to limit the scope of the Find operation to those files within the active project.

Display Options

Incremental search
This option causes the search process to begin as soon as a character is pressed, rather than waiting for OK to be pressed. When typing long search strings, you may find that the match is found before you're done, thereby saving typing. Just press *Enter* to dismiss the dialog and remain at the displayed match.


This option is disabled when Perl Regular Expressions are in use, since it is often the case that a regular expression cannot be properly evaluated until it has been completely typed.

Select matched text
This option causes the matched string to be selected so that it can be operated upon by any command that operates upon selected text. When this option is not used, the text cursor is simply placed at the start of the matching string.


This option is incompatible with the *Extend Selection* option, and will therefore be disabled when that checkbox is checked.

Highlight all matches
This option causes all instances of the matched string to be highlighted within the current file, and within other edited files. The highlighting will persist until a new Find operation is performed, or until the end of the editing session. Alternatively, highlighting can be disabled altogether using the View | Text Highlighting command (this also affects the general Text Highlighting feature). The foreground and background colors used to highlight matches can be set on the Configure | Colors dialog.


This option is not available for searches which use Regular Expressions.


To permanently configure one or more text strings for highlighting, use the Text Highlighting feature.

Extend Selection
This option can be used to extend an existing selection to the point of the matched string.


This option is logically incompatible with the *Select matched text* option, and will therefore be disabled when that checkbox is checked. This option will also be

disabled when the *Scope* has been set to *Selected text*.

Show at
This option controls the screen position at which matched strings are displayed. When a new match is already on-screen, it will be shown in place, without redrawing the screen. If the screen must be redrawn to show a new match, then the matching line will be positioned at the designated screen location.

Notes

When the *Incremental search* and *Select matched text* options are both in use, the Find Next and Find Previous commands can be issued from the keyboard (*F3* and *Shift+F3*, respectively) in order to display additional matches to the partially typed search string while the Find dialog is still open.

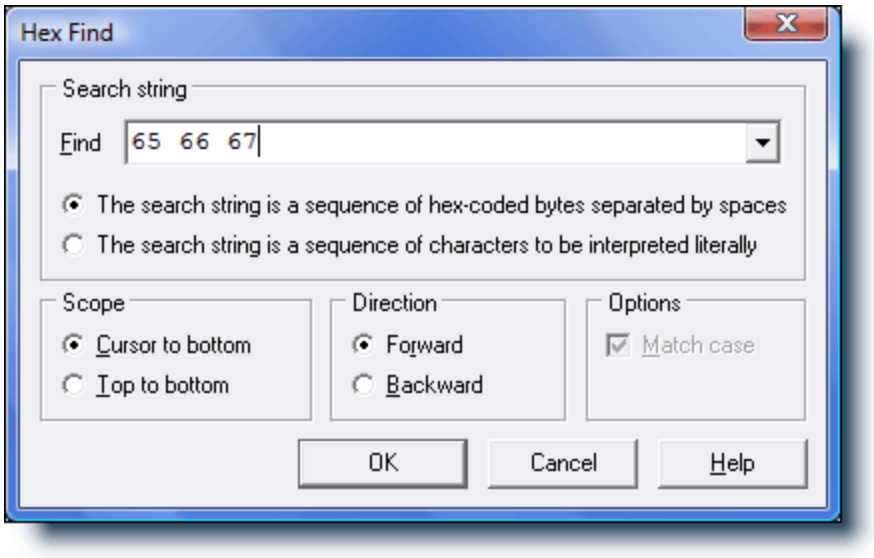
4.4.2 Find (Hex)

Menu: Search > Find

Default Shortcut Key: Ctrl+F

Macro function: Find()

When a file is being viewed or edited in hex mode, a special version of the Find dialog is presented for searching:



The Hex Find dialog has fewer options than the conventional Find dialog, and some extra options that are particular to hex searching.

Search String

Find
This is the edit box where the search string is entered. The search string can be entered as either a normal text string ("ABC," for example), or as a sequence of two-digit hex bytes separated by spaces ("41 42 43," for example). The radio buttons below the *Find* edit box can be used to force recognition of the search string in the format you intend, but in most cases they will sense what you're typing and adjust to the content of the string automatically.

Scope

Cursor to bottom / Cursor to top
This option causes the search to be performed from the text cursor onward, according to the current direction. The search ends when either the top or bottom of the file is reached.

Top to bottom / Bottom to top
This option causes the search to be performed from the top or bottom of file onward, according to the current direction.

Direction

Forward
This option causes the search to be performed downward, toward the end of file.

Backward
This option causes the search to be performed upward, toward the start of file.

Options

Match case
This option can be used to force the search string to be matched exactly. When unchecked, a case insensitive search is performed.

4.4.3 Find Next

Menu: Search > Find Next

Default Shortcut Key: F3

Macro function: FindNext()

The Find Next command is used to repeat the most recent search in a forward direction. The new search will obey all of the search options which were used when the search was first initiated with the Find command.


When the *Incremental search* and *Select matched text* options are both in use, the Find Next command can be issued from the keyboard in order to display additional matches to the partially typed search string while the Find dialog is still open.

4.4.4 Find Previous

Menu: Search > Find Previous

Default Shortcut Key: Shift+F3

Macro function: FindPrevious()

The Find Previous command is used to repeat the most recent search in a backward direction. The new search will obey all the of search options which were used when the search was first initiated with the Find command.

When the *Incremental search* and *Select matched text* options are both in use, the Find Previous command can be issued from the keyboard (*Shift+F3*) in order to display additional matches to the partially typed search string while the Find dialog is still open.

4.4.5 Find Fast

Menu: Search > Find Fast

Default Shortcut Key: Ctrl+F3

Macro function: FindFast()

The Find Fast command can be used to quickly search for the next occurrence of the word beneath the text cursor. The search is performed in the forward direction, toward the end of file. The search options from the Find command dialog box are used, even if the Find command has not yet been used in the current edit session.

4.4.6 Unhighlight Matches

Menu: Search > Unhighlight Matches

Default Shortcut Key: none

Macro function: UnhighlightMatches()

The Unhighlight Matches command removes on-screen highlighting from any text strings that matched the most recent Find operation.

The *Highlight all matches* option on the Find dialog causes matched strings to be highlighted throughout the current file. The Unhighlight Matches command removes that highlighting, without disabling the Find dialog option. When a new text string is searched for and found, it will again be highlighted. Put another way: this command provides a means to remove the highlighting added by the Find command without the need to disable the highlighting feature altogether, or perform a new, contrived search that is designed to fail.

4.4.7 Replace

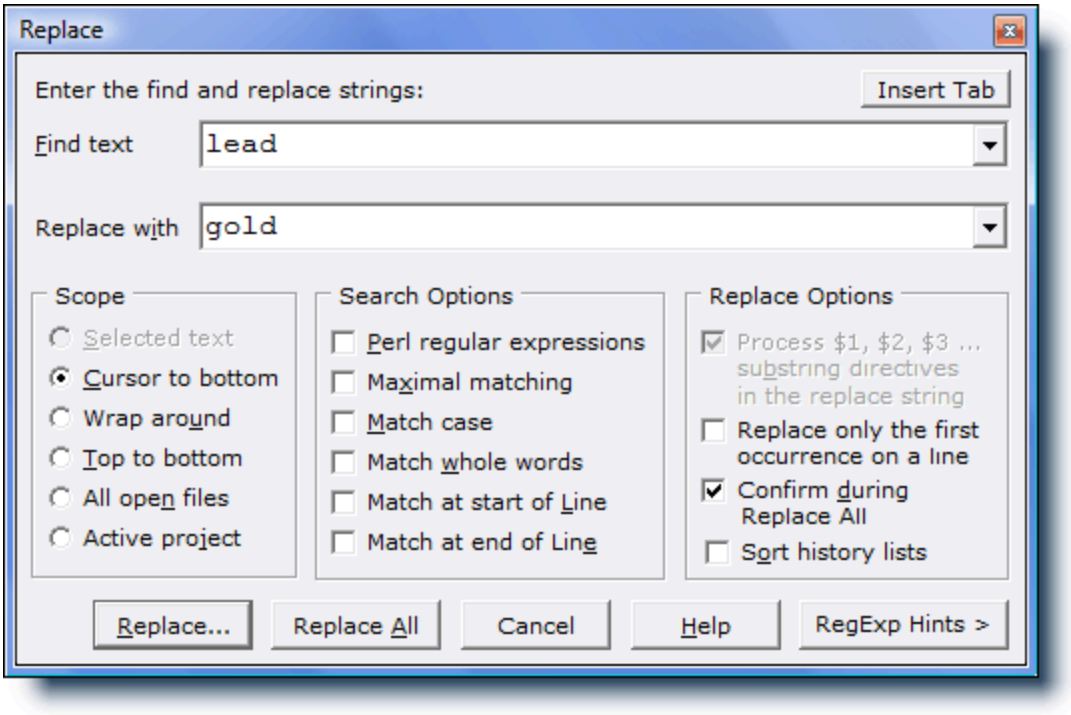
Menu: Search > Replace

Default Shortcut Key: Ctrl+R

Macro function: Replace()

The Replace command can be used to search for a text string and replace it with another string. Replacements can be made selectively or globally, within the current file or across all edited files. Regular Expressions can be entered within the search string.

The controls and options in the Replace dialog box are described below:



Find text
This is the edit box where the search string is entered. When the Replace command is issued, the word beneath the text cursor is placed into the *Find Text* edit box, in case that word--or a word which is nearly the same--is to be the search string. To recall a search string which was previously entered, use the drop-down list or press the up or down arrow keys to review the items in the history list. Regular Expressions may be used within the search string.

The *Delete* key can be used while the drop-down list is displayed to delete a selected entry from the history list.

Special characters can be entered into the *Find text* edit box using the technique

described in the Help topic Inserting Special Characters.

Replace with
This is the edit box where the replace string is entered. To recall a replace string which was previously entered, use the drop-down list or press the up or down arrow keys to review the items in the history list.


The *Delete* key can be used while the drop-down list is displayed to delete a selected entry from the history list.


The Replace command is line-oriented. It considers each line individually and does not look across line enders to match a search string which might span lines. Consequently, it is not possible to create new line enders using the Replace command, nor to delete existing line enders. For these types of operations, the Replace Line Enders command must be used.

Insert Tab
Use this button to insert a tab character into the *Find Text* or *Replace with* edit boxes.

Ordinarily, the *Tab* key is used to move from field to field within a dialog box. If you would prefer that the Tab key insert a tab character in this dialog box, and in other Find/Replace related dialog boxes, check the relevant box on the Configure | Preferences | Tabs dialog page.

Scope

Selected text
This option can be used to restrict the search and replace operation to the extent of the selected text.

Cursor to bottom
This option causes the search and replace operation to be performed from the cursor onward, toward the end of file. (There is no provision for making replacements in a backward direction.)

Wraparound
This option causes the search to be performed from the cursor onward, toward the end of file. When the end of file is reached, the search resumes at the top and continues to the original cursor position.

Top to bottom
This option causes the search and replace operation to be performed from the top of file onward, toward the end of file. (There is no provision for making replacements in a backward direction.)

All open files
This option causes the search and replace operation to be performed across all open files.

Active project
Use this option to limit the scope of the Replace operation to those files within the

active project.

Search Options

Perl regular expressions
If this box is checked, wildcard characters within the search string will be interpreted according to the Perl-Compatible Regular Expression (PCRE) convention. In part, this means that the asterisk (`*`) will cause a match of zero or more occurrences of the preceding character. The period (`.`) will match any single character. For more information, see Regular Expressions.

Maximal matching
When using pattern matching characters, there can sometimes be more than one text string that matches the search string. This option can be used to request that the longest possible matching string be returned.

Match case
This option can be used to force the search string to be matched exactly. When unchecked, a case insensitive search is performed.

Match whole words
This option can be used to restrict matches to those strings which appear as a whole word. The characters which serve to delimit words are user-configurable; see Configure | Preferences | Cursor.

Match at start of line
This option can be used to force the search string to be matched only when a matching string appears at the start of a line. This effect can also be achieved with a Regular Expression.

Match at end of line
This option can be used to force the search string to be matched only when a matching string appears at the end of a line. This effect can also be achieved with a Regular Expression.

Replace Options

Process $1, $2, $3... substring directives in the replace string
When this option is checked, special directives in the replace string will be replaced at match-time with subpatterns from the search string. This is a very powerful feature, as the following examples will illustrate.

Example 1:

Find text: `(\w+),(\w+)`
Replace with: `$2 $1`

The search string will match a string of one or more word characters followed by a comma, followed by another string of one or more word characters. For example: `Smith,John`. The parentheses are used to define subpatterns. The first open parenthesis indicates subpattern number 1, the next number 2, and so on. In this way,

the replace string can vary depending on what the search string matches. If the string `Smith,John` is matched, then the replace string will be `John Smith`. Running this search and replace operation on a data file would have the effect of inverting a list of `Lastname,Firstname` data to `Firstname Lastname` format.

Example 2:

Find text: `(Boxer|BOXER)`
Replace with: `<b>$1</b>`

This search string will match either `Boxer` or `BOXER`. The replace string will be equal to whatever the string matched, surrounded by the HTML open-bold and close-bold sequences. In this way, the target word can be replaced without regard to its case, while ensuring that no case conversion occurs due to the replacement.

☞ The entire matching string is designated as `$0`, even if subpatterns are not used. Up to 100 subpatterns can be referenced, numbering from `$0` to `$99`.

Example 3:

Find text: `"([^"]+),([^"]*)"`
Replace with: `"$1$2"`

This pair of search and replace strings can be used to remove commas from within the data fields of quote and comma-delimited data, without disturbing the commas that are used as field separators. The search pattern matches an entire double-quoted data field, so long as a comma appears within the data with at least one character to its left. The replace string references the data on either side of the comma as `$1` and `$2`, resulting in a replace string that duplicates the string matched, except that the comma is excluded. (If multiple commas appear within a single data field, you'll need to run the replace operation repeatedly until all occurrences have been replaced.)

☞ References to *named subpatterns* such as `(?P=name)` are also recognized in the replace string. See the Regular Expressions topic for more information about named subpatterns.

Replace only the first occurrence on a line
When this option is checked, only the first matching instance on a line will be considered eligible for replacement.

Confirm during Replace All
When this option is selected the *Replace All* operation will prompt before making each replacement. A dialog box will be presented so that each replacement can be confirmed. From this confirmation dialog box it is possible to later opt for unconditional replacements, by selecting its *All* button.

Sort history lists
If this box is checked the search and replace history lists will be maintained in alphabetic order, rather than in the order the strings were entered.

When switching to alphabetically sorted lists, the chronological ordering of the lists will be lost, and cannot be restored by unchecking the checkbox.

No attempt is made to associate the history list entries with the time that they were added to the list. If a sorted history list is used consistently, over time the list will come to hold an unrepresentative set of search phrases. In the extreme case, after many Replace operations, a list could result that contained only phrases beginning with the letter 'A'. This occurs because entries at the bottom of the list will be removed after the maximum size of the list is reached.

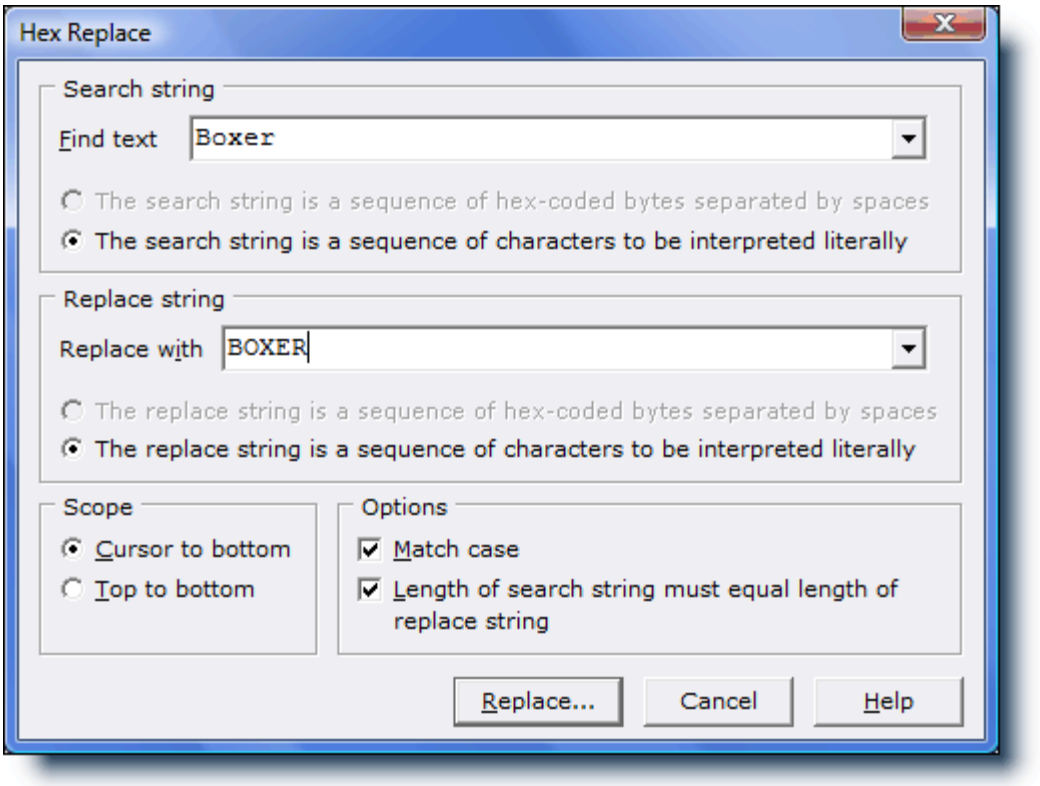
4.4.8 Replace (Hex)

Menu: Search > Replace

Default Shortcut Key: Ctrl+R

Macro function: Replace()

When a file is being edited in Hex Mode, a special version of the Replace dialog is presented for performing replacements:



The Hex Replace dialog has fewer options than the conventional Replace dialog, and some extra options that are particular to hex replacing.

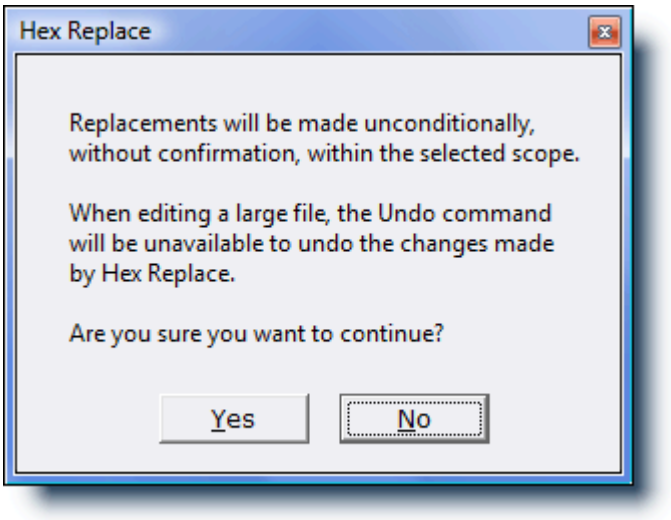
Search String

Find text
This is the edit box where the search string is entered. The search string can be entered as either a normal text string ("ABC," for example), or as a sequence of two-digit hex bytes separated by spaces ("41 42 43," for example). The radio buttons below the *Find text* edit box can be used to force recognition of the search string in the format you intend, but in most cases they will sense what you're typing and adjust to the content of the string automatically.

Replace String

Replace with
This is the edit box where the replacement string is entered. The replacement string can be entered as either a normal text string ("ABC," for example), or as a sequence of two-digit hex bytes separated by spaces ("41 42 43," for example). The radio buttons below the *Replace with* edit box can be used to force recognition of the replace string in the format you intend, but in most cases they will sense what you're typing and adjust to the content of the string automatically.

After the *Replace* button is clicked, a confirmation dialog appears:



As noted, the Replace Hex command does *not* provide an option to selectively choose which matches will be replaced. To begin the replace operation, click *Yes*.

Scope

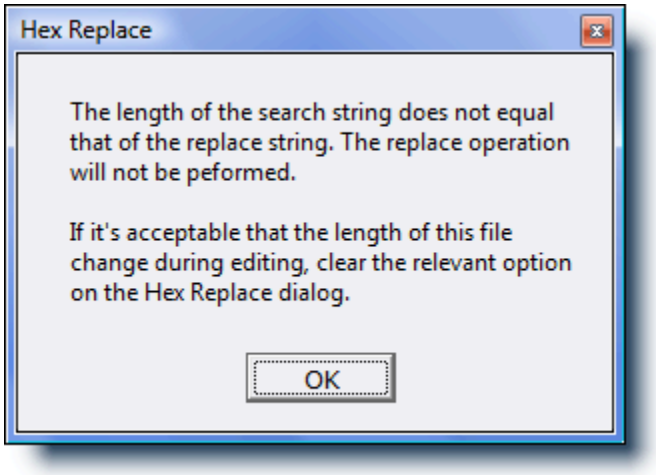
Cursor to bottom
This option causes the search and replace operation to be performed from the text cursor onward. The operation ends when the bottom of the file is reached, or when no additional matches can be found.

Top to bottom
This option causes the search and replace operation to be performed from the top of file onward. The operation ends when the bottom of the file is reached, or when no additional matches can be found.

Options

Match case
This option can be used to force the search string to be matched exactly. When unchecked, a case insensitive search is performed.

Length of search string must equal length of replace string
In many cases, it is not allowable to change the length of the hex/binary file that is being edited. When this option is checked, Boxer will enforce a requirement that the length of the search and replace string be equal, thereby guaranteeing that the file's length will not change due to a Replace operation. In this case, a warning dialog is presented:



4.4.9 Replace Again

Menu: Search > Replace Again

Default Shortcut Key: Shift+Ctrl+R

Macro function: ReplaceAgain()

The Replace Again command is used to repeat the most recent search and replace operation. The new operation will obey all of the options which were used when the previous search and replace operation was initiated with the Replace command.

4.4.10 Replace Line Enders

Menu: Search > Replace Line Enders

Default Shortcut Key: Ctrl+Alt+R

Macro function: ReplaceLineEnders()

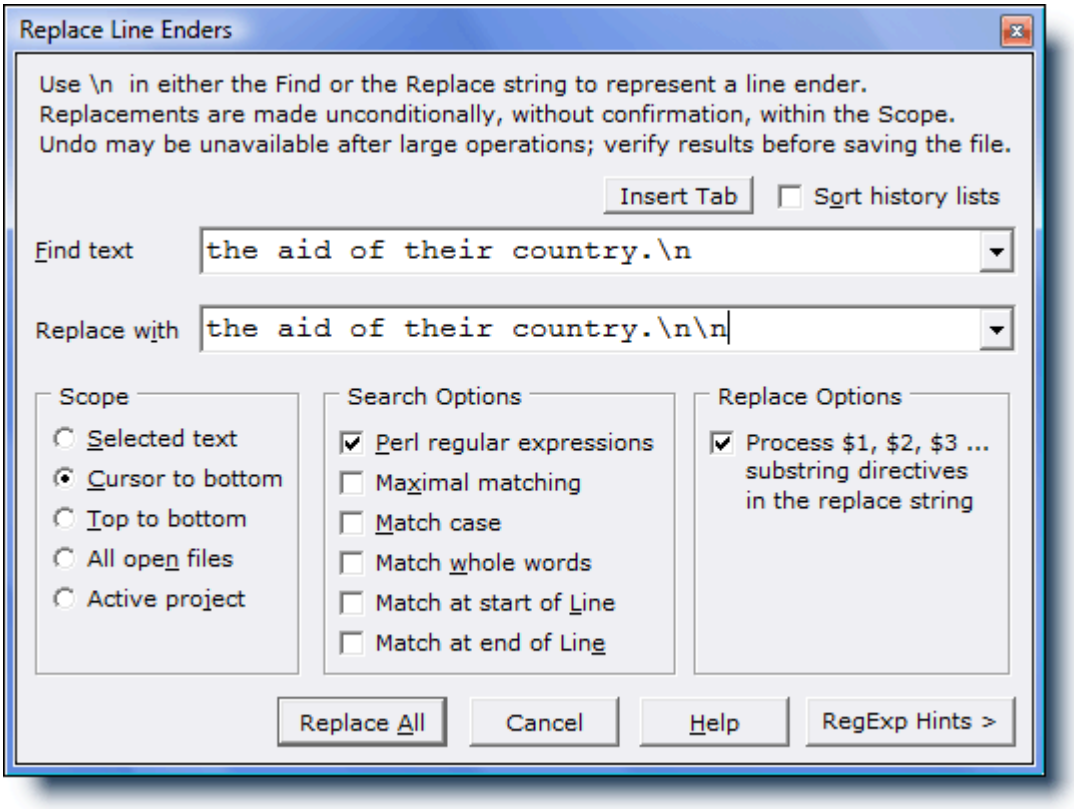
The Replace Line Enders command can be used to search for a text string and replace it

with another string. Replacements can be made selectively or globally, within the current file or across all edited files. Regular Expressions can be entered within the search string.

Unlike the Replace command, the Replace Line Enders command can be used to perform search and replace operations that span lines, and which may result in the addition or removal of lines from the file.

☞ Important Note: The Replace Line Enders command performs its replacements **unconditionally, without user confirmation.** When large operations are performed, Undo may be unavailable. For these reasons, it's advisable to **make a backup copy** of your file before using this command.

The controls and options in the Replace Line Enders dialog box are described below:



Find text
This is the edit box where the search string is entered. When the Replace Line Enders command is issued, the word beneath the text cursor is placed into the *Find Text* edit box, in case that word--or a word which is nearly the same--is to be the search string. To recall a search string which was previously entered, use the drop-down list or press the up or down arrow keys to review the items in the history list. Regular Expressions may be used within the search string.

Use the sequence \n to represent a line ender (newline). Example: if you are searching for a line that ends with 'jelly' that occurs just before a line that starts with 'bean', your

search string would be: `jelly\nbean`


The *Delete* key can be used while the drop-down list is displayed to delete a selected entry from the history list.


Special characters can be entered into the *Find text* edit box using the technique described in the Help topic Inserting Special Characters.

Replace with
This is the edit box where the replace string is entered. To recall a replace string which was previously entered, use the drop-down list or press the up or down arrow keys to review the items in the history list.

Use the sequence `\n` to represent a line ender (newline). Example: if you wanted to add two blank lines after all lines that end with the text `THE FOLLOWING:`, theses search and replace strings would be used:

Find text: `THE FOLLOWING:\n`
Replace with: `THE FOLLOWING:\n\n\n`


The *Delete* key can be used while the drop-down list is displayed to delete a selected entry from the history list.

Insert Tab
Use this button to insert a tab character into the *Find text* or *Replace with* edit boxes.

Ordinarily, the *Tab* key is used to move from field to field within a dialog box. If you would prefer that the Tab key insert a tab character in this dialog box, and in other Find/Replace related dialog boxes, check the relevant box on the Configure | Preferences | Tabs dialog page.

Scope

Selected text
This option can be used to restrict the search and replace operation to the extent of the selected text.

Cursor to bottom
This option causes the search and replace operation to be performed from the cursor onward, toward the end of file. (There is no provision for making replacements in a backward direction.)

Top to bottom
This option causes the search and replace operation to be performed from the top of file onward, toward the end of file. (There is no provision for making replacements in a backward direction.)

All open files

This option causes the search and replace operation to be performed across all open files.

Active project
Use this option to limit the scope of the Find operation to those files within the active project.

Search Options

Perl regular expressions
If this box is checked, wildcard characters within the search string will be interpreted according to the Perl-Compatible Regular Expression (PCRE) convention. In part, this means that the asterisk (`*`) will cause a match of zero or more occurrences of the preceding character. The period (`.`) will match any single character. For more information, see Regular Expressions.

Maximal matching
When using pattern matching characters, there can sometimes be more than one text string that matches the search string. This option can be used to request that the longest possible matching string be returned.

Match case
This option can be used to force the search string to be matched exactly. When unchecked, a case insensitive search is performed.

Match whole words
This option can be used to restrict matches to those strings which appear as a whole word. The characters which serve to delimit words are user-configurable; see Configure | Preferences | Cursor.

Match at start of line
This option can be used to force the search string to be matched only when a matching string appears at the start of a line. This effect can also be achieved with a Regular Expression.

Match at end of line
This option can be used to force the search string to be matched only when a matching string appears at the end of a line. This effect can also be achieved with a Regular Expression.

Replace Options

Process $1, $2, $3... substring directives in the replace string
When this option is checked, special directives in the replace string will be replaced at match-time with subpatterns from the search string. This is a very powerful feature, as the following examples will illustrate.

Example 1:

Find text: `(\w+),(\w+)`
Replace with: `$2 $1`

The search string will match a string of one or more word characters followed by a comma, followed by another string of one or more word characters. For example: `Smith,John`. The parentheses are used to define subpatterns. The first open parenthesis indicates subpattern number 1, the next number 2, and so on. In this way, the replace string can vary depending on what the search string matches. If the string `Smith,John` is matched, then the replace string will be `John Smith`. Running this search and replace operation on a data file would have the effect of inverting a list of `Lastname,Firstname` data to `Firstname Lastname`.

Example 2:

Find text: `(Boxer|BOXER)`
Replace with: `<b>$1</b>`

The search string will match either `Boxer` or `BOXER`. The replace string will be equal to whatever the string matched, surrounded by the HTML open-bold and close-bold sequences. In this way, the target word can be replaced without regard to its case, while ensuring that no case conversion occurs due to the replacement.

☞ The entire matching string is designated as $0, even if subpatterns are not used. Up to 100 subpatterns can be referenced, numbering from $0 to $99.

Sort history lists
If this box is checked the search and replace history lists will be maintained in alphabetic order, rather than in the order the strings were entered.

☞ When switching to alphabetically sorted lists, the chronological ordering of the lists will be lost, and cannot be restored by unchecking the checkbox.

☞ No attempt is made to associate the history list entries with the time that they were added to the list. If a sorted history list is used consistently, over time the list will come to hold an unrepresentative set of search phrases. In the extreme case, after many Replace operations, a list could result that contained only phrases beginning with the letter 'A'. This occurs because entries at the bottom of the list will be removed after the maximum size of the list is reached.

4.4.11 Find Mate

Menu: Search > Find Mate

Default Shortcut Key: Ctrl+]

Macro function: FindMate()

The Find Mate command locates the mating parenthetical element to the parenthetical element at the text cursor, and moves the text cursor ahead (or back) to that position. The search begins at the text cursor and will continue all the way to the start or end of the file, as may be needed.

If the text cursor is sitting on an opening parenthetical character such as `(`, `[`, `<`, or `{`, the cursor will be moved ahead to the corresponding closing mate, with consideration given to nesting. If the cursor is situated on a closing parenthetical character such as `)`, `]`, `>`, or `}`, the cursor will be moved backward to the corresponding opening mate, again with consideration given to nesting.

The Find Mate command also recognizes text strings as parenthetical elements, and many of the most common parenthetical pairs have been pre-defined. For example: if the cursor is sitting on `begin`, Find Mate will locate `end`. If the cursor is sitting on `<i>` (the HTML code to begin italics), Find Mate will find `</i>`. If the cursor is sitting on `while`, Find Mate will find `endwhile`.

The Find Mate command can be used to extend an existing text selection to a closing element. For example, to select a parenthesized block of text, select the opening parenthesis and issue the Find Mate command. The selection will be extended to include all of the text up to and including the closing parenthesis.

The parenthetical pairs recognized by Find Mate can be viewed and/or defined on the Configure | Preferences | Editing 1 options page. The name of the option is *Set mating pairs for Find Mate*.


When editing a file for which syntax highlighting information is available, Find Mate will ignore parenthetical elements that occur within block comments, end-of-line comments, character constants and quoted strings. If syntax highlighting is disabled, or unavailable, this feature cannot be performed.


Find Mate can also be used to test for unmated parenthetical elements, since a request to find a mate for an unbalanced element will result in a report that the closing mate could not be found.


When defining Find Mate pairs for tagged languages such as HTML, remember that commands which include parameters will need different treatment than commands that cannot use parameters. For example, if you were to define 'table' using the definition `<table>=</table>`, Boxer would not be able to find matches when 'table' was used with parameters, such as `<table width="200">`. For this reason, a definition of the form `<table=</table>` should be used instead, without the closing `>` character.


Find Mate is not able to handle mating pairs whose beginning or ending element is shared by other parenthetical pairs. For example, the definitions `#if=#endif`, `#ifdef=#endif` and `#ifndef=#endif` all share the same closing element, `#endif`. The nesting complexities that could arise from such definitions is beyond the scope of the Find Mate command.

4.4.12 Find and Count

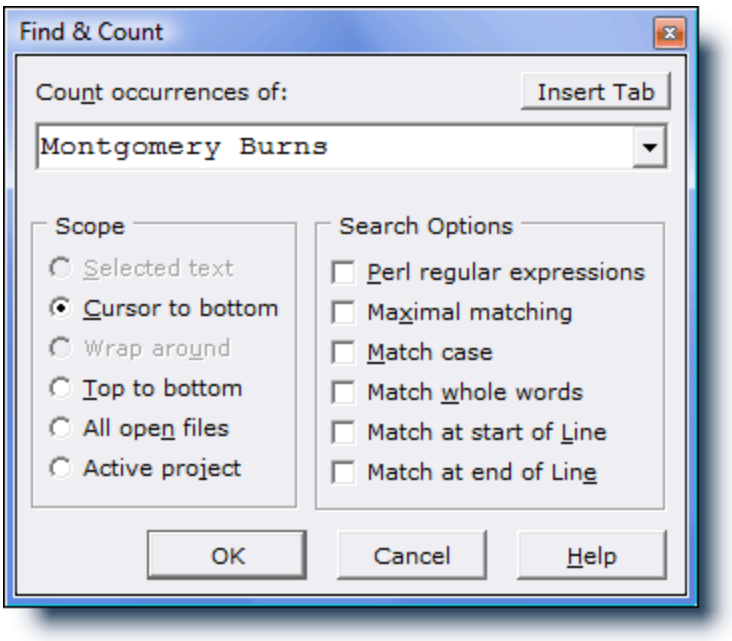
Menu: Search > Find and Count

Default Shortcut Key: none

Macro function: FindAndCount()

The Find and Count command can be used to count the number of occurrences of a specified text string within the current file, or within all edited files. Find and Count is a passive operation, it simply reports the number of matches found for the specified string, within the specified range.

The search options that appear in the Find and Count dialog box are the same as those which appear in the Replace dialog box. See the Replace topic for full details.



The Find and Count command reports its result using a read-only edit box so that the value can be copied to the Windows clipboard.

4.4.13 Find a Disk File

Menu: Search > Find a Disk File

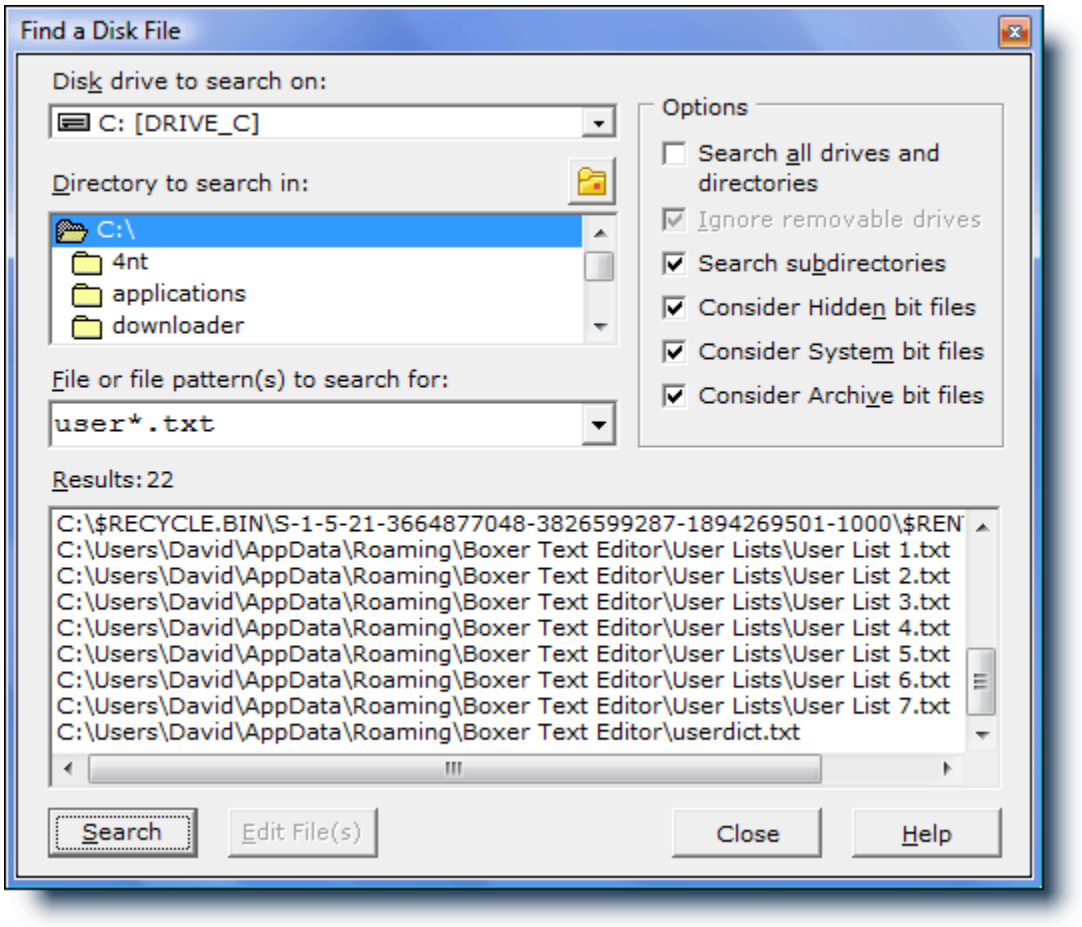
Default Shortcut Key: none

Macro function: FindADiskFile()

The Find a Disk File command provides the ability to locate one or more disk files which match a supplied filename or file pattern(s). Matching filenames are displayed in a results window, and one or more files from the list can then be selected for editing. The dialog box is *non-modal* and stay-on-top, so you can peruse the files opened and later return to the dialog to open other files, without losing the search results.

A range of options are provided to control how the search is conducted, and what

drives, directories and files should be considered:



Disk drive to search on
This drop-down list allows you to specify the disk drive to be searched. When the *Search all drives and directories* checkbox is selected, this list is disabled.

Directory to search in
This drop-down list allows you to specify the directory to be searched. When the *Search all drives and directories* checkbox is selected, this list is disabled.

Use the folder icon with the red square in it to jump to the directory of the current file.

File or file pattern(s) to search for
This edit box allows you to specify the filename and/or file pattern(s) to search for. A list of common file patterns has been supplied in the drop-down list, or you can type your own. When specifying multiple patterns, separate the patterns with a semi-colon and do not use intervening spaces.

Options

Search all drives and directories
If this option is selected all drives and subdirectories will be searched, except that removable drives may be exempted using the option below.

Ignore removable drives
If this option is selected drives with removable media (such as floppy drives and mass storage cartridges) will not be searched.

Search subdirectories
If this option is selected all subdirectories below the selected directory will also be searched.

Consider Hidden bit files
If this option is selected, files whose Hidden attribute bit is set will be considered during the file search. The Hidden attribute causes a file to become invisible to many directory listing programs, and is often used together with the System file attribute.

Consider System bit files
If this option is selected, files whose System attribute bit is set will be considered during the file search. The System attribute is sometimes used by the operating system to distinguish files which should not be altered or deleted.

Consider Archive bit files

If this option is selected, files whose Archive attribute bit is set will be considered during the file search. The Archive attribute is used by the operating system to flag those files which have been changed since the previous backup operation. Backup programs will typically reset a file's Archive bit after saving the file to a backup device. In most cases you will want to leave this checkbox active to ensure that recently changed files will be searched.

4.4.14 Find Text in Disk Files

Menu: Search > Find Text in Disk Files

Default Shortcut Key: none

Macro function: FindTextInDiskFiles()

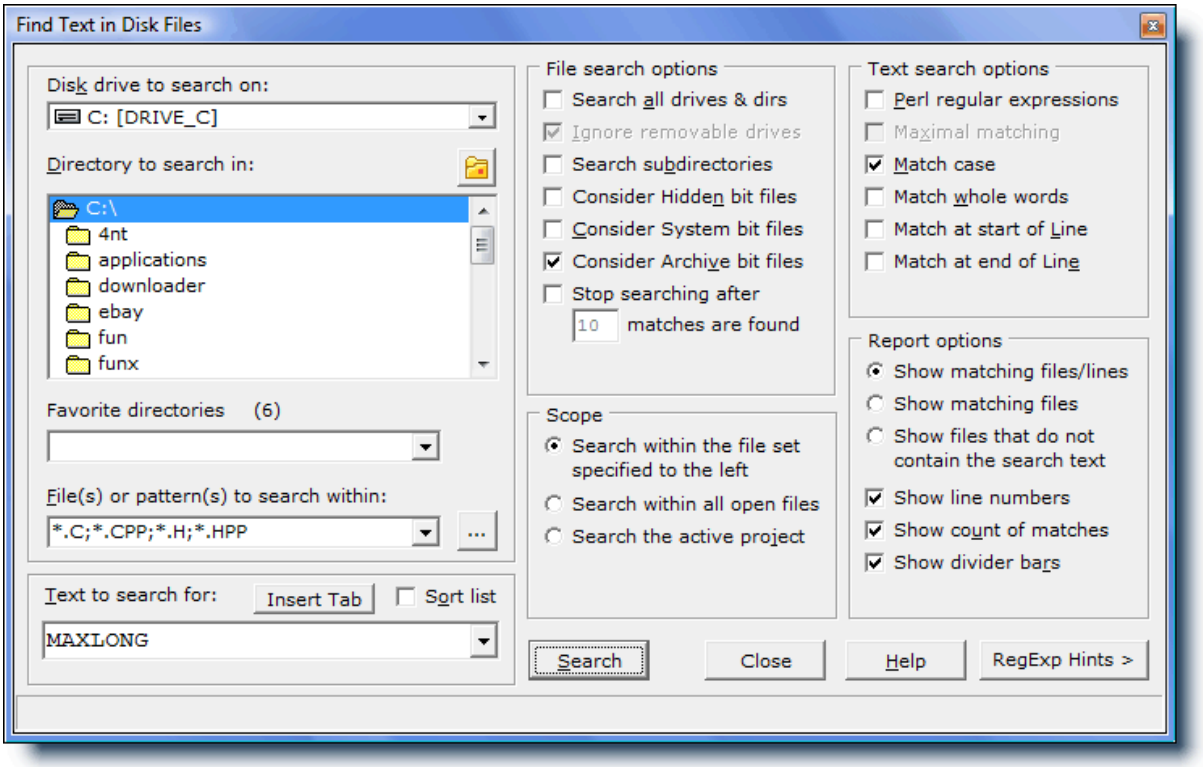
The Find Text in Disk Files command provides the ability to search for a text string across a specified range of drives, directories and files. Lines which contain the desired string are presented in a results window, and the file containing a match can be opened by pressing *Enter* or double clicking on the line.

Regular Expressions can be used when specifying the search string, and one or more file patterns can be used to search within an entire class of files.

The results window is *non-modal*, so you can peruse the files opened and later return to the window to open other files, without losing the search results. The results window has a *Copy All* button which allows its results to be copied to the current clipboard. The *Copy Selected* button will copy only those lines that have been selected. The *Open*

All button will automatically open all files in which matching lines were found.

A wide range of options are provided to control how the search is conducted, and what drives, directories and files should be searched:



Disk drive to search on
This drop-down list allows you to specify the disk drive to be searched. When the *Search all drives and directories* checkbox is selected, this list is disabled.

Directory to search in
This drop-down list allows you to specify the directory to be searched. When the *Search all drives and directories* checkbox is selected, this list is disabled. Note that a double-click is required to select a directory; a single-click will not suffice.

Use the folder icon with the red square in it to jump to the directory of the current file.

Favorite directories
This control can be used to recall other directories that have been used in the past.

File(s) or file pattern(s) to search within
This edit box allows you to specify the filename and/or file pattern(s) to search within. A list of common file patterns has been supplied in the drop-down list, or you can type your own. When specifying multiple patterns, separate the patterns with a semi-colon (;) and do not use intervening spaces.

The file patterns that appear in the drop-down list are shared with the File Open dialog. The file patterns which appear in this dialog are user-definable via the Configure | Preferences | File I/O options page.

Regardless of whether or not such files match the supplied filename(s) or file pattern(s), binary files will not be searched by this command.

Text to search for
This edit box is used to specify the text string to be found. Regular Expressions can used if desired. The associated drop-down list can be used to recall previous search strings.

The *Delete* key can be used while the drop-down list is displayed to delete the selected entry from the history list.

Special characters can be entered into the *Text to search for* edit box using the technique described in the Help topic Inserting Special Characters.

Insert Tab
Use this button to insert a tab character into the *Text to search for* edit box.

Ordinarily, the *Tab* key is used to move from field to field within a dialog box. If you would prefer that the Tab key insert a tab character in this dialog box, and in other Find/Replace related dialog boxes, check the relevant box on the Configure | Preferences | Tabs dialog page.

Sort List
If this box is checked the history list will be maintained in alphabetic order, rather than in the order the strings were entered.

When switching to an alphabetically sorted list, the chronological ordering of the list will be lost, and cannot be restored by unchecking the checkbox.

File Search Options

Search all drives and directories
If this option is selected all drives and subdirectories will be searched, except that removable drives may be exempted using the option below.

Ignore removable drives
If this option is selected drives with removable media (such as floppy drives and mass storage cartridges) will not be searched.

Search subdirectories
If this option is selected all subdirectories below the selected directory will also be searched.

Consider Hidden bit files
If this option is selected, files whose Hidden attribute bit is set will be considered during the search. The Hidden attribute causes a file to become invisible to many directory listing programs, and is often used together with the System file attribute.

Consider System bit files
If this option is selected, files whose System attribute bit is set will be considered during the search. The System attribute is sometimes used by the operating system to distinguish files which should not be altered or deleted.

Consider Archive bit files
If this option is selected, files whose Archive attribute bit is set will be considered during the search. The Archive attribute is used by the operating system to flag those files which have been changed since the previous backup operation. Backup programs will typically reset a file's Archive bit after saving the file to a backup device. In most cases you will want to leave this checkbox active to ensure that recently changed files will be searched.

Stop searching after *n* matches are found
Use this option to stop the search after a specified number of matches have been found.

Scope

Search within the file set specified to the left
Use this option to search a file set which has been designated in the disk, directory and file controls at the left side of the dialog.

Search within all open files
Use this option to restrict the search to those files that are currently open for editing.

Search the active project
Use this option to limit the scope of the Find operation to those files within the active project.

Text Search options

Perl regular expressions
If this box is checked, wildcard characters within the search string will be interpreted according to the Perl-Compatible Regular Expression (PCRE) convention. In part, this means that the asterisk (*) will cause a match of zero or more occurrences of the preceding character. The period (.) will match any single character. For more information, see Regular Expressions.

Maximal matching
When using pattern matching characters, there can sometimes be more than one text string that matches the search string. This option can be used to request that the longest possible matching string be returned.

Match case
This option can be used to force the search string to be matched exactly. When unchecked, a case insensitive search is performed.

Match whole words
This option can be used to restrict matches to those strings which appear as a whole

word. The characters which serve to delimit words are user-configurable; see Configure | Preferences | Cursor.

Match at start of line
This option can be used to force the search string to be matched only when a matching string appears at the start of a line. This effect can also be achieved with a Regular Expression.

Match at end of line
This option can be used to force the search string to be matched only when a matching string appears at the end of a line. This effect can also be achieved with a Regular Expression.

Report Options

Show matching files/lines
If this option is selected, the text of the matching lines will be shown in the results window, grouped by file.

Show matching files
If this option is selected, the filenames in which matching lines occurred will be reported, but the matching lines will not be shown..

Show files that do not contain the search text
If this option is selected, the filenames in which matching lines do *not* appear will be reported..

Show line numbers
If this option is selected, line numbers will be displayed to the left of each matching line.

Show count of matches
If this option is selected, the count of matches found in each file will be shown in the results window.

Show divider bars

If this option is selected, divider bars will be used within the results window to separate one file's matches from another.


If you prefer that the Find Text in Disk Files dialog automatically start in the directory of the current file, use the relevant option on the Configure | Preferences | File I/O dialog page.

4.4.15 Find Duplicate Lines

Menu: Search > Find Duplicate Lines

Default Shortcut Key: none

Macro function: FindDuplicateLines()

The Find Duplicate Lines command can be used to locate all lines within the current file which are duplicated elsewhere in the file. The duplicate lines are copied, with line numbers, into an untitled file. No change is made to the current file during the operation.

If a range of lines is selected, Find Duplicate Lines will operate only on that portion of the file.

The results are presented in alphabetic order. The Sort Lines command can be used to sort by line numbers, if desired.

If you need to delete duplicate lines, use the Delete Duplicate Lines command.


This command can be useful for finding duplicate items within a list which is expected to contain only unique entries. For example: given a list of charitable donor names, Find Duplicate Lines could be used to find those parties who have contributed more than once.

4.4.16 Find Unique Lines

Menu: Search > Find Unique Lines

Default Shortcut Key: none

Macro function: FindUniqueLines()

The Find Unique Lines command can be used to locate all lines within the current file which are *not* duplicated elsewhere in the file. The unique lines are copied, with line numbers, into an untitled file. No change is made to the current file during the operation.

If a range of lines is selected, Find Unique Lines will operate only on that portion of the file.

The results are presented in alphabetic order. The Sort Lines command can be used to sort by line numbers, if desired.

The effect of this command is similar to the Find Distinct Lines command, with an important difference: Find Unique Lines omits any lines which are duplicated from its report. Find Distinct Lines includes duplicated lines, but places just a single instance of such lines in its report. An example will clarify:

Original File's Content...	Find Unique Lines gives...	Find Distinct Lines gives...
AAA	AAA	AAA
BBB	DDD	BBB
BBB	EEE	CCC

CCC	FFF	DDD
CCC		EEE
DDD		FFF
EEE		
FFF		


This command can be useful for finding items within a list which are not duplicated elsewhere in the list. For example: given a list of zip codes to which deliveries must be made, Find Unique Lines could be used to find those zip codes for which only one delivery must be made, allowing special arrangements to be made.

4.4.17 Find Distinct Lines

Menu: Search > Find Distinct Lines

Default Shortcut Key: none

Macro function: FindDistinctLines()

The Find Distinct Lines command can be used to isolate all distinct lines within the current file. The distinct lines are copied, with line numbers, into an untitled file. No change is made to the current file during the operation.

If a range of lines is selected, Find Distinct Lines will operate only on that portion of the file.

The results are presented in alphabetic order. The Sort Lines command can be used to sort by line numbers, if desired.

The effect of this command is similar to the Find Unique Lines command, with an important difference: Find Unique Lines omits from its report any lines which are duplicated. Find Distinct Lines includes duplicated lines, but places just a single instance of such lines in its report. An example will clarify:

Original File's Content...	Find Unique Lines gives...	Find Distinct Lines gives...
AAA	AAA	AAA
BBB	DDD	BBB
BBB	EEE	CCC
CCC	FFF	DDD
CCC		EEE
DDD		FFF
EEE		

FFF		

This command can be useful for isolating distinct entries in a list. For example: a file contains lists of email address that were merged from multiple sources. Find Distinct Lines could be used to create a new list that contains one occurrence of each distinct email address. (Following the same example, a similar result could be obtained using the Delete Duplicate Lines command.)

4.4.18 Find Differing Lines

Menu: Search > Find Differing Lines

Default Shortcut Key: Ctrl+D

Macro function: FindDifferingLines()

The Find Differing Lines command can be used to locate differing lines among two or more similar files. After the command is issued, the text cursor will be advanced in each open file to the next line whose text is not identical in among all open files.

This command will be most efficient when used as follows: open the files to be compared and select Window | Tile Across or Window | Tile Down to arrange the windows in a left-to-right or top-to-bottom arrangement. Position the text cursor in each file to line 1, or to a line just before where the comparison is to begin. *In any case, the text on each starting line should be identical among all the files being compared*. When the Find Differing Lines command is issued, the text cursor will be advanced in each file until a line is found which differs among the open files.

The first differing column in the line will be highlighted, and the operation is complete. You can make any corrections that might be needed and then issue the command again to find the next difference. If the difference that was found involves the addition or deletion of one or more lines, the files will need to be re-synchronized manually before proceeding. That is, the text cursor must be moved in each file to a line with identical content so that a new comparison can begin.

Find Differing Lines ignores minimized files during its operation, so if there are any files open which should *not* be compared they can be minimized before beginning.

4.5 Jump Menu

4.5.1 Go to Line

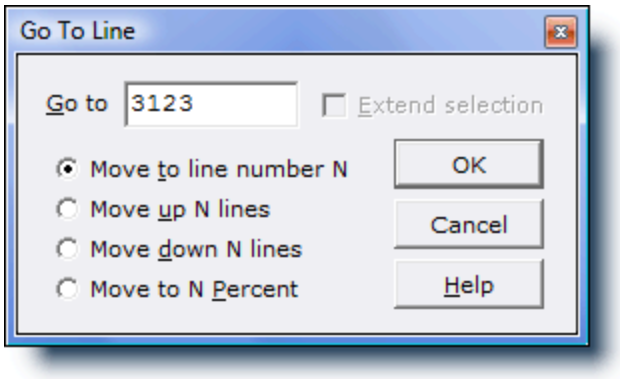
Menu: Jump > Go to Line

Default Shortcut Key: Ctrl+G

Macro function: GoToLine()

The Go to Line command can be used to jump immediately to a specified line number in the current file. Options are also provided in the Go to Line dialog box to move up or down by the value specified, or to treat the value as a percentage. For example, specifying 50% would result in movement to a line midway through the current file.

If text is selected when this command is issued, an option will be available to extend the selection to the new location.



The current line number is always displayed on the Status Bar, next to the 'L' label. The Go to Line command can also be issued by double clicking within the line number display in the Status Bar.

The Go To Line number dialog also recognizes the following syntax: +10 to jump ahead 10 lines; -15 to jump back 15 lines, and 45% to move to the 45 percent position in the file. The use of this syntax overrides the mode indicated by the radiobutton options.

4.5.2 Go to Column

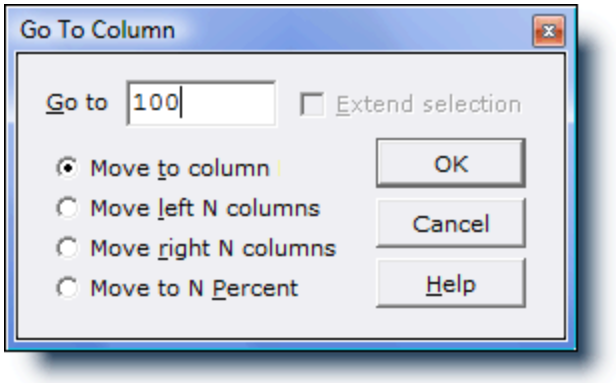
Menu: Jump > Go to Column

Default Shortcut Key: Shift+Ctrl+G

Macro function: GoToColumn()

The Go to Column command can be used to jump immediately to a specified column number on the current line. Options are also provided in the Go to Column dialog box to move left or right by the value specified, or to treat the value as a percentage. For example, specifying 25% would result in movement to column 25 in a line with 100 characters.

If text is selected when this command is issued, an option will be available to extend the selection to the new location.



The current column number is always displayed on the Status Bar, next to the 'C' label. The Go to Column command can also be issued by double clicking within the column number display in the Status Bar.

The Go To Column dialog also recognizes the following syntax: +10 to jump ahead 10 columns; -15 to jump back 15 columns, and 45% to move to the 45 percent position along the current line. The use of this syntax overrides the mode indicated by the radiobutton options.

4.5.3 Go to Byte Offset

Menu: Jump > Byte Offset

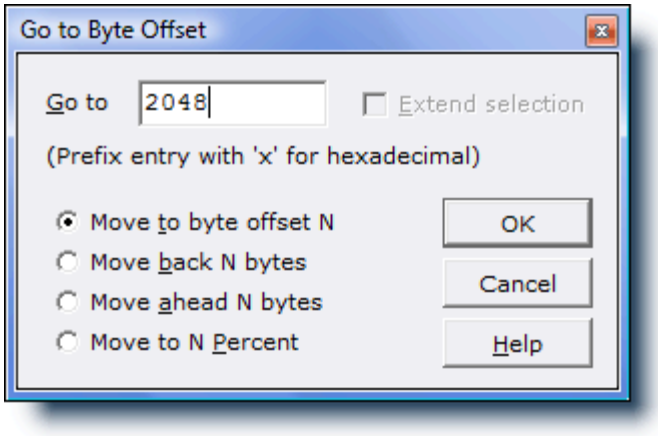
Default Shortcut Key: none

Macro function: GoToByteOffset()

The Go to Offset command can be used to jump immediately to a specified byte offset within the current file. Options are also provided in the Go to Offset dialog box to move backward or forward by the value specified, or to treat the value as a percentage. For example, specifying 50% would result in movement to a character midway through the file.

When viewing a file in Hex Mode, the Go to Byte Offset command will adjust itself to provide the expected movement to positions within the hex display. A *hexadecimal* offset can be specified by prefixing the value entered with an 'x'.

If text is selected when this command is issued, an option will be available to extend the selection to the new location.



The Go To Offset dialog also recognizes the following syntax: +10 to jump ahead 10 bytes; -15 to jump back 15 bytes, and 45% to move to the 45 percent position in the file. The use of this syntax overrides the mode indicated by the radiobutton options.

4.5.4 Next Bookmark

Menu: Jump > Next Bookmark

Default Shortcut Key: Shift+Ctrl+Down

Macro function: NextBookmark()

The Next Bookmark command moves the text cursor to the nearest bookmark which appears below the text cursor's current location. If the Next Bookmark command finds no bookmarks below the current line, the text cursor will wrap around and be placed on the first bookmark in the file.

Travel among bookmarks is based upon location, not bookmark number.

The Bookmark Manager can be used to view all bookmarked lines in a single view, and navigate to, or delete, selected bookmarks.

Bookmarks will persist for the current editing session, and will be restored when restoring an edit session.

If a selection exists when this command is issued, the selection will be extended to the bookmarked location.

4.5.5 Previous Bookmark

Menu: Jump > Previous Bookmark

Default Shortcut Key: Shift+Ctrl+Up

Macro function: PreviousBookmark()

The Previous Bookmark command moves the text cursor to the nearest bookmark which appears above the text cursor's current location. If the Previous Bookmark command finds no bookmarks above the current line, the text cursor will wrap around and be placed on the last bookmark in the file.

Travel among bookmarks is based upon location, not bookmark number.

The Bookmark Manager can be used to view all bookmarked lines in a single view, and navigate to, or delete, selected bookmarks.

Bookmarks will persist for the current editing session, and will be restored when restoring an edit session.


If a selection exists when this command is issued, the selection will be extended to the bookmarked location.

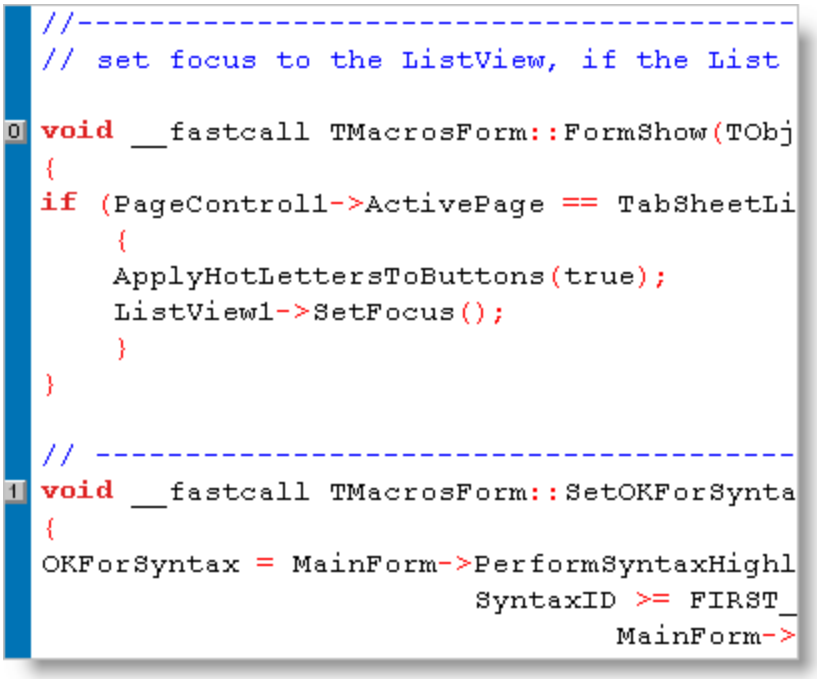
4.5.6 Toggle Bookmark

Menu: Jump > Toggle Bookmark

Default Shortcut Key: F9

Macro function: ToggleBookmark()

The Toggle Bookmark command places a bookmark at the current location of the text cursor, or clears a bookmark if the line is already bookmarked. Bookmarks are displayed at the far left edge of the window as a small number (0-9) within a gray box. Up to ten bookmarks can be placed in a file at any one time.



```
//---------------------------------------
// set focus to the ListView, if the List

void __fastcall TMacrosForm::FormShow(TObj
{
if (PageControl1->ActivePage == TabSheetLi
    {
    ApplyHotLettersToButtons(true);
    ListView1->SetFocus();
    }
}

// ---------------------------------------
void __fastcall TMacrosForm::SetOKForSynta
{
OKForSyntax = MainForm->PerformSyntaxHighl
                        SyntaxID >= FIRST_
                                MainForm->
```

If the bookmarked column is altered due to the addition or deletion of text on the bookmarked line, the bookmark will be adjusted automatically. If a line containing a bookmark is deleted, the bookmark will be cleared.

Bookmarks can be used to mark various points of interest within a text file. Once one or more lines have been bookmarked, you can use the Previous Bookmark and Next Bookmark commands to move among the bookmarked lines. The Bookmark Manager can be used to view all bookmarked lines in a single view, and navigate to, or delete, selected bookmarks.

Bookmarks will persist for the current editing session, and will be restored when restoring an edit session.

The display of bookmarks is controlled by the View | Bookmarks command. Bookmarks remain operational even if they are not currently being shown on-screen.

4.5.7 Bookmark Manager

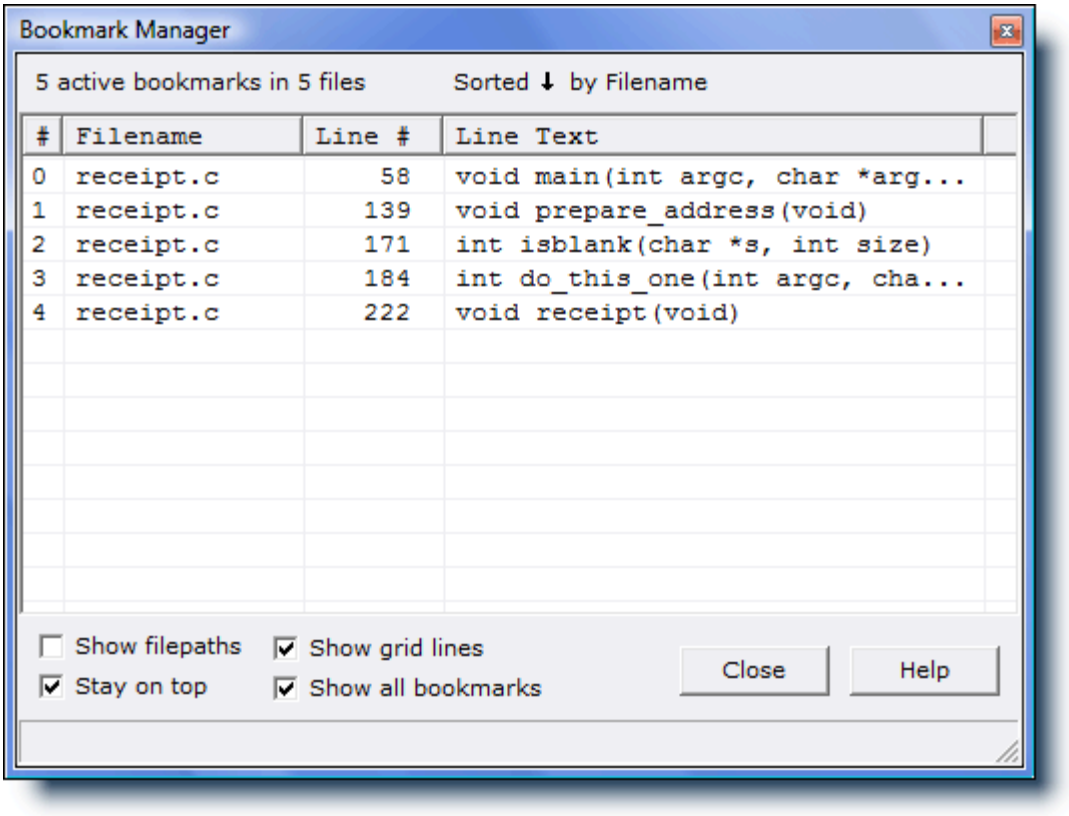
Menu: Jump > Bookmark Manager

Default Shortcut Key: Shift+F9

Macro function: BookmarkManager()

The Bookmark Manager command displays a pop-up dialog showing all bookmarked lines in the files being edited. Double clicking on an entry causes the associated file to become current, and the cursor to be placed on the bookmarked line. The display can be sorted on any of the fields by clicking on the header bar at the top of each field. Press the *Delete* key to remove a bookmark.

The Bookmark Manager can be left open while working in Boxer, so that it's available for reference or quick navigation.



Show filepaths
Use this option to control whether filenames or full filepaths are displayed for each bookmark entry.

Stay on Top
This checkbox controls whether or not the dialog will remain on top of other windows.

Show Grid Lines
Use this option to toggle on/off the display of grid lines within the view.

Show all bookmarks
This checkbox can be used to control whether bookmarks are displayed for all open files, or for only the current file.

Bookmarks will persist for the current editing session, and will be restored when restoring an edit session.

4.5.8 Next Paragraph

Menu: Jump > Next Paragraph

Default Shortcut Key: none

Macro function: NextParagraph()

The Next Paragraph command moves the text cursor to the start of the next paragraph.

For purposes of this command, a paragraph is considered to be a block of lines with one or more empty lines between them. Contiguous paragraphs which are denoted by a change of indent on the first line, and not by an intervening blank line, will not be recognized to be distinct paragraphs.

If a text selection is present when this command is issued, the selection will be extended to the new cursor location.

4.5.9 Previous Paragraph

Menu: Jump > Previous Paragraph

Default Shortcut Key: none

Macro function: PreviousParagraph()

The Previous Paragraph command moves the text cursor to the start of the previous paragraph.

For purposes of this command, a paragraph is considered to be a block of lines with one or more empty lines between them. Contiguous paragraphs which are denoted by a change of indent on the first line, and not by an intervening blank line, will not be recognized to be distinct paragraphs.

If a text selection is present when this command is issued, the selection will be extended to the new cursor location.

4.5.10 Go to Paragraph

Menu: Jump > Go to Paragraph

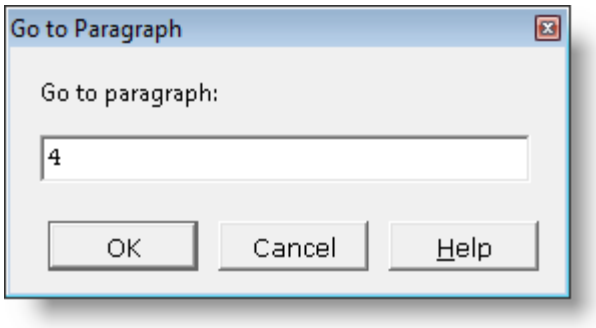
Default Shortcut Key: none

Macro function: GoToParagraph()

The Go to Paragraph command can be used to jump immediately to a specified paragraph number in the current file. The distinction between lines and paragraphs relates to the Visual Wrap feature. When Visual Wrap is active, lines with soft line

enders are wrapped to width of the window, or to some other wrapping margin. In Visual Wrap mode, a single physical line of text might occupy more than one line on the screen; screen line 11 might correspond to paragraph 4.

Go to Paragraph can be used to move easily among paragraphs:



The distinction between lines and paragraphs will become more obvious if the View Line Numbers option is active.

4.5.11 Next Function

Menu: Jump > Next Function

Default Shortcut Key: Ctrl+Alt+Down

Macro function: NextFunction()

The Next Function command moves the cursor to the next function (or procedure) declaration within the current file. This command makes it possible to move through a source code file on a function-by-function basis.

If a text selection is present when this command is issued, the selection will be extended to the new cursor location.

The Next Function command relies upon the Ctags Function Index feature to perform its service. If the Ctags feature has been disabled, or if the file being edited is not supported by Ctags, the Next Function command will be unavailable.

The types of Ctags identifiers for which this command applies is user-configurable. The default setting includes entries for "function", "procedure", "subroutine", "method", etc. The full list can be viewed or changed on the *Advanced* tab of the Configure | Ctags Function Indexing dialog.

4.5.12 Previous Function

Menu: Jump > Previous Function

Default Shortcut Key: Ctrl+Alt+Up

Macro function: PreviousFunction()

The Previous Function command moves the cursor to the previous function (or procedure) declaration within the current file. This command makes it possible to move backward through a source code file on a function-by-function basis.


If a text selection is present when this command is issued, the selection will be extended to the new cursor location.


The Previous Function command relies upon the Ctags Function Index feature to perform its service. If the Ctags feature has been disabled, or if the file being edited is not supported by Ctags, the Previous Function command will be unavailable.


The types of Ctags identifiers for which this command applies is user-configurable. The default setting includes entries for "function", "procedure", "subroutine", "method", etc. The full list can be viewed or changed on the *Advanced* tab of the Configure | Ctags Function Indexing dialog.

4.5.13 Declaration

Menu: Jump > Declaration

Default Shortcut Key: none

Macro function: Declaration()

When editing within a supported source code file, the Declaration command provides a means to jump from an identifier reference to the point at which the identifier was declared. For example, when sitting at a function/procedure call, issuing the Declaration command will cause the cursor to jump to the declaration of the function/procedure being referenced. The Declaration command can also be used to jump to the declaration point of defined constants and global variables, so long as these entities are indexed by Ctags.

If the declaration resides in another file, that file will be opened and/or made current before moving the cursor to the relevant declaration line. If more than one declaration exists for the identifier at the cursor, the Ctags Function Index dialog will be displayed so that the proper instance can be selected.


The Declaration command relies upon the Ctags Function Index feature to perform its service. If the Ctags feature has been disabled, or if the file being edited is not supported by Ctags, the Declaration command will be unavailable.


After moving to a declaration, use the Reference command to return to the identifier reference from which the Declaration command was made.

4.5.14 Reference

Menu: Jump > Reference

Default Shortcut Key: none

Macro function: Reference()

The Reference command is used in conjunction with the Declaration command. After issuing the Declaration command to jump from an identifier reference to its declaration, use the Reference command to return to the point of reference.

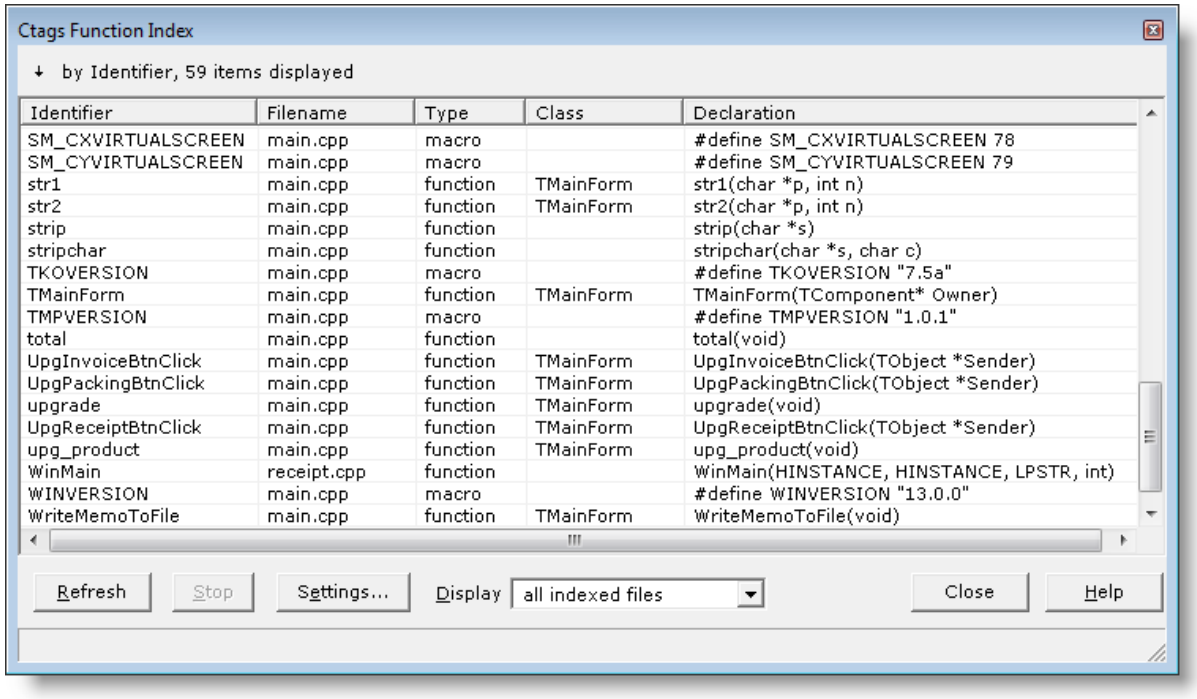
4.5.15 Ctags Function Index

Menu: Jump > Ctags Function Index

Default Shortcut Key: none

Macro function: CtagsFunctionIndex()

The Ctags Function Index command displays a dialog containing a list of functions, procedures and global variables for the files currently being edited. The list can be used as a handy reference to function names and their calling parameters, or as a navigation aid: double-clicking on an entry will jump to the file and line that corresponds to the highlighted entry. The dialog is *non-modal*, so it can remain open alongside Boxer as you're doing other work.



In order to index the edited files, Boxer runs an external program and then reads the output file it creates. Exuberant Ctags is a fast, multi-language implementation of the original *ctags* and *etags* programs that are available on Unix. Exuberant Ctags is

distributed under the *GNU General Public License.* The program `ctags.exe` and a zip file containing the program's source code have been installed in a directory named 'Ctags' beneath the Boxer installation directory.

Exuberant Ctags provides built-in support for indexing the following languages:

Ant	HTML	Ruby
Assembler	Jave	Scheme
ASP	Javascript	Shell scripts (Bourne/Korn/Z)
Awk	Lisp	S-Lang
Basic	Lua	SML (Standard ML)
BETA	Make	Tcl
C and C++	MATLAB	TeX
C#	Objective Caml	Vera
COBOL	Pascal	Veilog
DOS Batch	Perl	VHDL
Eiffel	PHP	Vim
Erlang	PL/SQL	YACC
Flex	Python	
Fortran	REXX	

In addition, Boxer is supplied with a `CTAGS.CNF` configuration file that adds support for these languages:

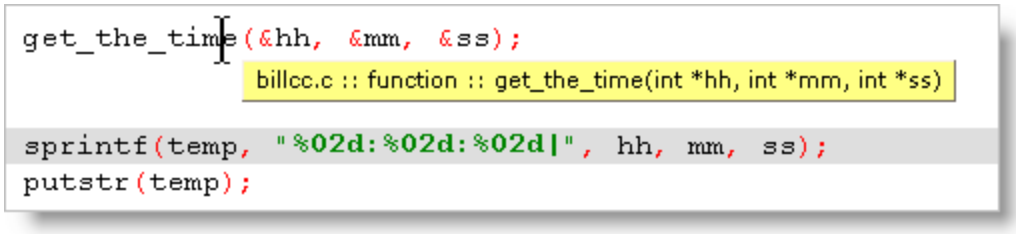
ActionScript	Latex	System Verilog
Cascading Style Sheets	Miva	XML
INI files		


Support for indexing additional languages can be added by making additions to the `CTAGS.CNF` file. The process is not trivial, however, and it's often easier to find a configuration on the internet which has been developed by someone else. Instructions for adding additional languages can be found at the Exuberant Ctags website. By keeping your copy of Ctags up-to-date, you can also be assured of getting access to new built-in languages as they are added by its developers. The version of Ctags that was supplied with Boxer was current at the time of Boxer's release.

The list can be sorted on any of its columns by clicking on the associated column title in the header at the top of the listing. Clicking on the same header a second time will reverse the order of the sort.

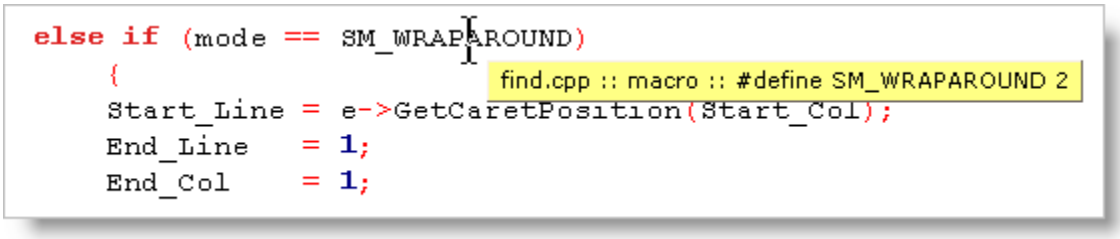
The function prototype information contained in the Ctags Function Index dialog is also

available for display when the mouse hovers over a function that has been indexed:



The display of these popup tool tips can be configured by clicking the Settings button, which leads to the Configure Ctags Function Indexing dialog.

Popup tool tips can also be displayed for global variables, structure and class members, typedefs, macros and other language-dependent identifiers:



Refresh
Use the *Refresh* button to re-index all open files, and any other 'extra files' that may have been designated in the Configure Ctags Function Indexing dialog. You might want to use the *Refresh* button when changes have been made to an edited file that would invalidate the information that was previously gathered. For example, if a function's calling parameters are changed, or a function is added or deleted, use *Refresh*.

It is **not** necessary to use *Refresh* simply because the line number of a function's declaration has changed. The indexing is maintained in a format that is not sensitive to changes in line numbers.

Settings
The Settings button will display the Configure Ctags Function Indexing dialog, which provides options that control how and when files will be indexed, the appearance of the function list, and whether popup tool tips will be displayed.

Display
Use the *Display* combobox to filter the listing of indexed functions and variables. The available choices are:

all indexed files
all open files
files in active project
current file
extra files

Extra files to be indexed can be designated in the Configure Ctags Function Indexing dialog.

☞ The *Display* setting will also influence which identifiers are visible to the popup tool tip feature. If you have filtered the listing to show only those entries in the *current file*, for example, you may wish to restore the *Display* setting to *all indexed files* so that the full collection of indexed identifiers are available to the popup tool tips feature.

4.5.16 Make Line Top

Menu: Jump > Make Line Top

Default Shortcut Key: none

Macro function: MakeLineTop()

The Make Line Top command causes the screen to be redrawn so that the current line is at the top of screen. This command is useful for showing the text below the current view without losing the current line.

If there is insufficient text to fill the window, the command will be disabled.

4.5.17 Make Line Center

Menu: Jump > Make Line Center

Default Shortcut Key: none

Macro function: MakeLineCenter()

The Make Line Center command causes the screen to be redrawn so that the current line is at the middle of the screen.

4.5.18 Make Line Bottom

Menu: Jump > Make Line Bottom

Default Shortcut Key: none

Macro function: MakeLineBottom()

The Make Line Bottom command causes the screen to be redrawn so that the current line is at screen bottom. This command is useful for showing the text above the current view without losing the current line.

If there is insufficient text to fill the window, the command will be disabled.

4.5.19 Left Window Edge

Menu: Jump > Left Window Edge

Default Shortcut Key: none

Macro function: LeftWindowEdge()

The Left Window Edge command positions the text cursor to the left edge of the current window. If the file has been scrolled to the right, the amount of scroll will not be affected.

If a text selection is present when this command is issued, the selection will be extended to the new cursor location.

4.5.20 Right Window Edge

Menu: Jump > Right Window Edge

Default Shortcut Key: none

Macro function: RightWindowEdge()

The Right Window Edge command positions the text cursor to the right edge of the current window.

If the cursor has been constrained to move from the end of a line to the start of the next line, the behavior of the Right Window Edge command will be impacted. In this case, when issued on a line that does not reach the right window edge, this command will move to the cursor to the end of the line.

If a text selection is present when this command is issued, the selection will be extended to the new cursor location.

4.5.21 Backtab

Menu: Jump > Backtab

Default Shortcut Key: Alt+Left Arrow

Macro function: Backtab()

The Backtab command is used to move the text cursor backward to the previous tabstop. The size of tabstops is determined by the Tab Display Size command.

4.6 Paragraph Menu

4.6.1 Visual Wrap

Menu: Paragraph > Visual Wrap

Default Shortcut Key: Alt+F10

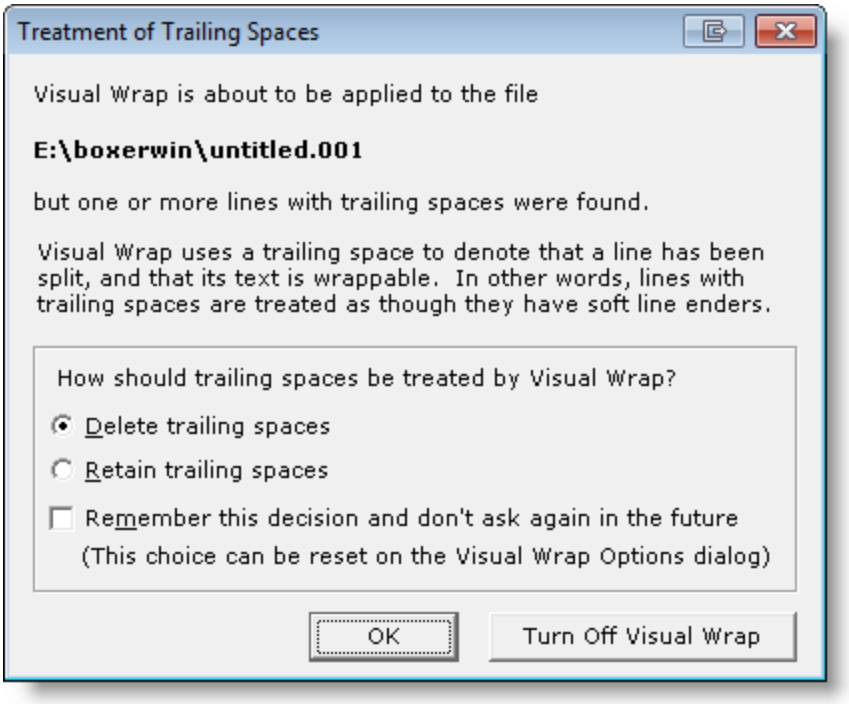
Macro function: VisualWrap()

The Visual Wrap command toggles a passive display mode that causes long lines to be wrapped to the window width (or another wrapping margin) without introducing hard line enders into the file, as would occur if the Reformat command were used. Visual Wrap is useful when editing files with very long lines that would otherwise extend off-screen to the right, out of view. It's also useful for preparing text that will later be imported into another program that prefers a long, flowing stream of text without intra-paragraph line enders.

When Visual Wrap is active, the line counter in the status bar switches to a paragraph counter, since a one-to-one relationship between screen lines and physical lines no longer exists. If Line Numbers are being displayed in the left margin, that display is also adjusted to show paragraph numbers rather than line numbers. When Visible Spaces are in use, the soft line enders on wrapped line will be denoted with a double chevron symbol (<<), while hard line enders are marked with a single chevron (<). The Go to Paragraph command can be used to jump to a paragraph by its number.

The Visual Wrap Options dialog provides access to options related to the operation of Visual Wrap. Wrapping can be set to occur at the window width, the Text Width, or at the Right Margin Rule. By default, Visual Wrap is maintained when edits are made, although this can be optionally disabled. An option is also provided for dealing with trailing spaces when Visual Wrap is first applied.

Boxer uses a trailing space to mark lines that are split or wrapped by Visual Wrap. When Visual Wrap is first applied to a file, a check is performed to see if any lines in the file already contain trailing spaces. If such lines are found, the following dialog is presented:



The nature of the file being processed will determine which option should be selected. If the trailing spaces are extraneous, the first option (delete) should be used. If the file is one in which trailing spaces are being used to mark soft line enders, the second option should be used.

☞ The Soften Line Enders command can be used to prepare a file for processing by Visual Wrap. It converts hard line enders to soft line enders, with proper consideration to paragraph boundaries, thereby making the lines of a text file flowable.

☞ The Harden Line Enders command can be used convert soft line enders to hard line enders, thereby making permanent the current on-screen formatting.

Visual Wrap Theory

When Visual Wrap is activated, any lines that are longer than the designated wrapping margin are wrapped to fit within the margin. A trailing space is left at the location where each line was split to denote that the line has a "soft" line ender. Some word processing programs insert a special character into the data stream to denote a soft line ender. As a text editor, Boxer is obliged not to introduce special characters into the text files it creates, so it simply adds a space to the end of the line. The last line of a paragraph ends with a hard line ender, and thus has no trailing space.

The use of a trailing space to mark lines with a soft line ender has several advantages:

- The space character can be seen when Visible Spaces mode is active.
- The space character will not be visible when the document is printed.
- Line enders can be easily converted between hard and soft by adding or removing the space.

☞ **Note**: When Visual Wrap is applied to (or removed from) an entire file, the length of the text on each line changes, as does the total number of lines. The text itself is not changed, just its on-screen formatting. But internally, these changes cause any previously stored Undo information to become invalid. Ordinarily, Visual Wrap will be enabled just after a file is opened. If the wrapping margin is not changed thereafter, Visual Wrap will have very little impact on Undo. But if a file is frequently rewrapped or unwrapped, this side effect is something to bear in mind.

☞ Visual Wrap mode is incompatible with Typing Wrap. If Typing Wrap is on when Visual Wrap is enabled, it will be automatically turned off.

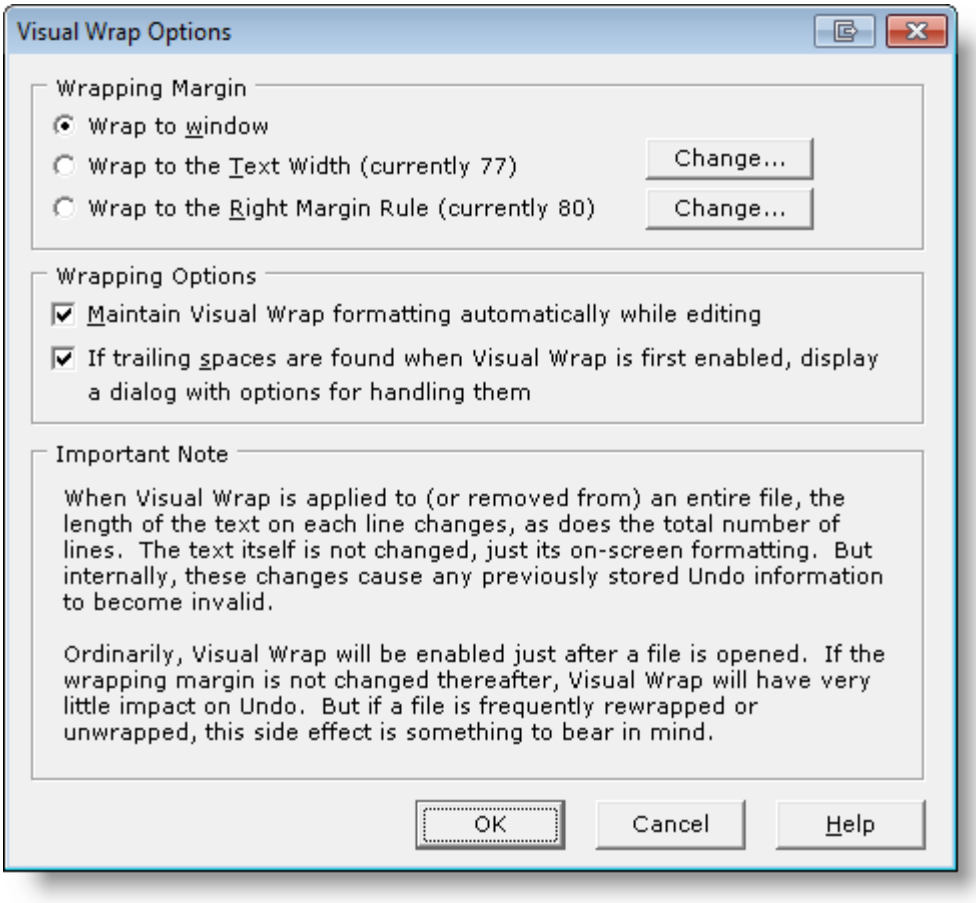
4.6.2 Visual Wrap Options

Menu: Paragraph > Visual Wrap Options

Default Shortcut Key: none

Macro function: VisualWrapOptions()

The Visual Wrap Options dialog contains settings that relate to the operation of Visual Wrap:



Wrapping Margin

Wrap to window
Choose this option if you want text to be wrapped to the width of the document window. The text will re-wrap automatically if the window is resized.

Wrap to the Text Width
Choose this option if you want text to be wrapped to the Text Width value that's used for various paragraph operations, such as Reformat. When this option is used, text will only rewrap when the Text Width value is changed, and not when the document window is simply resized.

Wrap to the Right Margin Rule
Choose this option if you want text to be wrapped to the Right Margin Rule value. The Right Margin Rule is an optional display feature that causes a fine vertical line to appear on-screen at the designated column. When this option is used, text will only rewrap when the Right Margin Rule value is changed, and not when the document window is simply resized.

Wrapping Options

Maintain Visual Wrap formatting automatically while editing

This is option is enabled by default. You might choose to disable this option if you find yourself distracted by the automatic wrapping that occurs when editing which causes lines to split and join automatically.

If trailing spaces are found when Visual Wrap is first enabled, display a dialog with options for handling them
When trailing spaces are found in a file to which Visual Wrap is being applied, Boxer displays a dialog with options for how these spaces should be handled. On that dialog, and option appears to disable display of the dialog. This option provides a means to re-enable the display of that dialog.

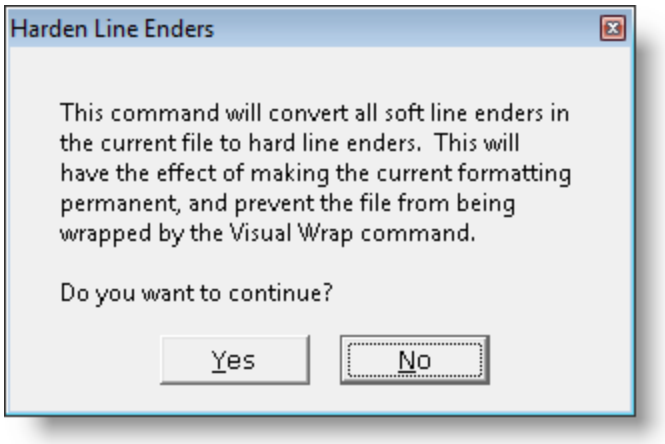
4.6.3 Harden Line Enders

Menu: Paragraph > Harden Line Enders

Default Shortcut Key: none

Macro function: HardenLineEnders()

The Harden Line Enders command converts soft line enders to hard line enders. If a selection is present, the operation is restricted to the selected range of lines. If a selection is not present, the operation is performed across the whole file. A confirmation dialog will appear before the operation is performed:



The concept of "soft" and "hard" line enders relates to the Visual Wrap command. A line with one or more spaces at the end is considered to have a soft line ender. Lines without trailing spaces are considered to have hard line enders. When Visual Wrap mode is active, lines with soft line enders are eligible to be merged with the content of lines below, allowing text to be reformatted to fit within the window width (or whatever other wrapping margin is chosen).

Applying the Harden Line Enders command to a file has the effect of making the current on-screen formatting permanent... until or unless the Soften Line Enders command is used to reverse this operation. If you apply the Harden Line Enders command to a

selected range of lines, these lines will be ineligible for wrapping by the Visual Wrap command.

See also: Visual Wrap, Visual Wrap Options, Soften Line Enders

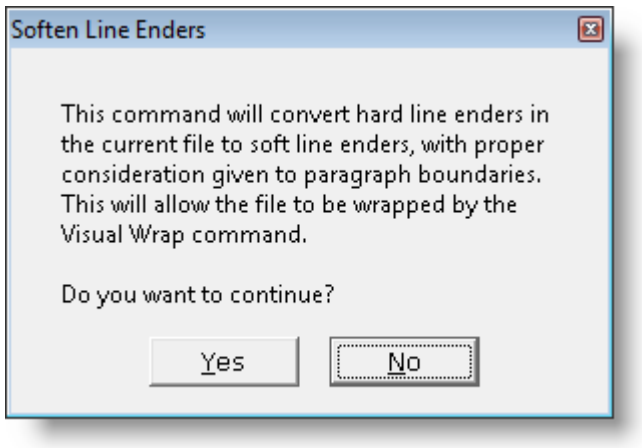
4.6.4 Soften Line Enders

Menu: Paragraph > Soften Line Enders

Default Shortcut Key: none

Macro function: SoftenLineEnders()

The Soften Line Enders command converts hard line enders to soft line enders, with proper consideration to paragraph boundaries. If a selection is present, the operation is restricted to the selected range of lines. If a selection is not present, the operation is performed across the whole file. A confirmation dialog will appear before the operation is performed:



The concept of "soft" and "hard" line enders relates to the Visual Wrap command. A line with one or more spaces at the end is considered to have a soft line ender. Lines without trailing spaces are considered to have hard line enders. When Visual Wrap mode is active, lines with soft line enders are eligible to be merged with the content of lines below, allowing text to be reformatted to fit within the window width (or whatever other wrapping margin is chosen).

Applying the Soften Line Enders command to a file has the effect of making the file flowable by Visual Wrap.

See also: Visual Wrap, Visual Wrap Options, Harden Line Enders

4.6.5 Reformat

Menu: Paragraph > Reformat

Default Shortcut Key: Ctrl+F10

Macro function: Reformat()

The Reformat command can be used to reformat the paragraph at the text cursor within the defined Text Width and according to the current Justification Style. The Reformat operation begins on the current line and includes all lines to the end of the current paragraph (see note below). The text cursor is advanced to the first line of the next paragraph following Reformat, so that successive Reformat commands will move smoothly through the document.

If a range of lines is selected, all paragraphs within the selected range will be reformatted. Use the Select All Text command before Reformat to reformat an entire file, but first check to be sure that the file doesn't contain tables or lists which might be adversely affected by reformatting.


When the Reformat command reformats text, it does so by adding a newline (hard line ender) at the end of the line. To wrap text visually, without introducing a hard line ender into the file, see the Visual Wrap command.

Fully Indented Paragraphs
Boxer uses the amount of indent on the second line of the paragraph to determine the indent level for the entire paragraph. A paragraph can be made *fully indented* by manually indenting the first and second lines of the paragraph and then reformatting.

```
   This paragraph is fully indented. This
   paragraph is fully indented. This paragraph
   is fully indented. This paragraph is fully
   indented. This paragraph is fully indented.
   This paragraph is fully indented.
```

Hanging Indents
The indent on the first line of the paragraph is not applied to other lines in the paragraph. A *hanging indent* can be achieved by placing less indent on the first line of the paragraph than on the second line. Likewise, if the first line of a paragraph has extra indent, it too will be preserved.

```
This paragraph has a hanging indent. This paragraph
   has a hanging indent. This paragraph has a
   hanging indent. This paragraph has a hanging
   indent. This paragraph has a hanging indent. This
   paragraph has a hanging indent.
```

Bulleted Paragraphs
If you wish to create bulleted paragraphs that will retain their layout after being reformatted, use a Tab character to separate the bullet from the body of the paragraph. If a series of Spaces were to be used between the bullet and the body text they would be adjusted during Reformat. Tabs are maintained in this situation.

```
*  This paragraph uses a bullet separated from the
   body text with a Tab character. This paragraph
   uses a bullet separate from the body text with a
```

```
Tab character.
```

To have text wrap to the next line automatically as you type, use the Typing Wrap feature. To wrap text visually, use the Visual Wrap command.

The Unformat command is essentially the opposite of Reformat: it removes line enders from a paragraph to create a long, flowing line of text.

The end of a paragraph is signaled by the presence of one or more blank lines between paragraphs, but *not* simply by the presence of additional indent on a line which immediately follows the current paragraph. Lines which begin with a period (`.`) will also be recognized as blank lines for purposes of Reformat. This is to permit the use of text markup languages such as Flexicon for adding formatting commands to text.

Reformat will break lines between HTML or XML tags, when appropriate, even if an intervening space is not present.

4.6.6 Unformat

Menu: Paragraph > Unformat

Default Shortcut Key: Ctrl+Alt+F10

Macro function: Unformat()

The Unformat command can be used to convert the lines of the current paragraph into a single, long line. The Unformat operation begins on the current line and includes all lines to the end of the current paragraph. The text cursor is advanced to the first line of the next paragraph following Unformat, so that successive Unformat commands will move smoothly through the document.

If a range of lines is selected, all paragraphs within the selected range will be processed. Use the Select All Text command before Unformat to process an entire file, but first check to be sure that the file doesn't contain tables or lists which would be adversely affected by the new formatting.

See also the Soften Line Enders command.

If the total length of the paragraph being unformatted is greater than the maximum line length (see Sizes and Limits), the operation will use multiple lines.

The Unformat command is useful for preparing text that is to be imported into a word processor, email client or other programs that perform 'soft formatting'. Programs such as these sometimes require that extra newlines be removed before imported text can be properly formatted.

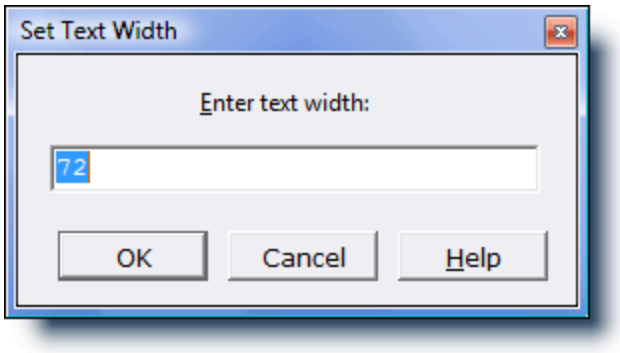
4.6.7 Text Width

Menu: Paragraph > Text Width

Default Shortcut Key: Ctrl+W

Macro function: TextWidth()

The Text Width command is used to set the column at which text justification commands will wrap words to the next line. A popup dialog box will appear to retrieve a new value for the Text Width:



The Text Width value is used by the following commands during their operation:

Typing Wrap
Reformat
Quote and Reformat
Align Left
Align Center
Align Right
Align Smooth

The current Text Width is displayed on the Status Bar, next to the 'w' indicator which displays Typing Wrap status. The Text Width command can also be issued by double clicking on the Text Width value in the Status Bar.

The maximum value for the Text Width command is 9999.

4.6.8 Justification Style

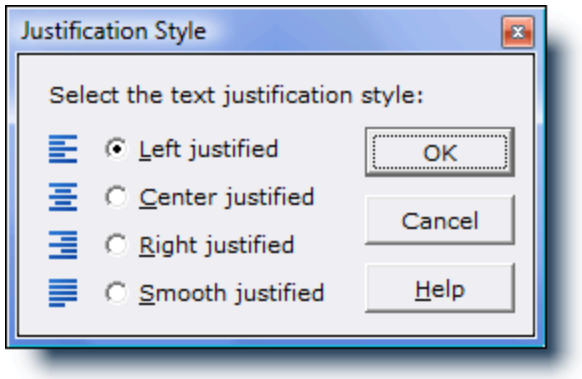
Menu: Paragraph > Justification Style

Default Shortcut Key: Ctrl+J

Macro function: JustificationStyle()

The Justification Style command is used to set the justification style used by the Reformat, Typing Wrap and Quote and Reformat commands. There are four justification

styles to choose from:



The paragraphs below show examples of each justification style:

Left Justified - text will be justified flush against the left edge, with the right edge being left ragged.

Center Justified - text will be centered within the current text width, with the left and right edges being ragged.

Right Justified - text will be justified flush against the right edge, with the left edge being left ragged.

Smooth Justified - text will be flush against both the left and right margins. Spaces are inserted alternately in the left, center, and right portions of a line to minimize the appearance of 'rivers and valleys' in the justified text.

4.6.9 Typing Wrap

Menu: Paragraph > Typing Wrap

Default Shortcut Key: Ctrl+F5

Macro function: TypingWrap()

The Typing Wrap command is used to toggle on and off typing wrap mode. When Typing Wrap mode is on, text typed from the keyboard will be wrapped to the next line when the Text Width value is exceeded. When Typing Wrap mode is off, typed text will not be wrapped to the next line until the *Enter* key is pressed.

☞ When Boxer wraps text to the next line in Typing Wrap mode, it does so by adding a true newline (hard line ender) character at the end of the line. To wrap text visually, without introducing a hard line ender into the file, see the Visual Wrap command.

Typing Wrap mode is maintained separately for each edited file. Activating Typing Wrap in one file does not affect the Typing Wrap mode for other edited files.

The current file's Typing Wrap mode is displayed on the Status Bar. An uppercase 'W' indicates that Typing Wrap is on. A lowercase 'w' indicates that Typing Wrap is off. The Typing Wrap command can also be issued by double clicking on the 'w' value in the Status Bar.

☞ See also the Visual Wrap command.

☞ Typing Wrap will break lines between HTML or XML tags, when appropriate, even if an intervening space is not present.

☞ Prior to Boxer v14, the Typing Wrap command was called Word Wrap. When the Visual Wrap command was added, the command was changed to Typing Wrap for clarity.

4.6.10 Quote and Reformat

Menu: Paragraph > Quote and Reformat

Default Shortcut Key: Ctrl+Q

Macro function: QuoteAndReformat()

The Quote and Reformat command can be used to reformat a paragraph within the defined Text Width and according to the current Justification Style, while adding a quoting symbol to the left edge of the paragraph. This formatting style is used within email replies and in other communications to visually identify the text which is being replied *to* from the text of the reply itself.

Two quoting styles are available: one in which the first line is quoted and additional lines are indented to match the first line:

```
>> A  Multi-User  License  provides an inexpensive
   way for businesses,  schools,  universities  or
   other  work  groups  to  supply their personnel
   with computer software in both a legal and cost
   efficient manner.  By  licensing  Boxer for use
   on multiple computers you can standardize on  a
   single  editing  tool that will serve the needs
   of all people within the group.
```

and one in which all lines within the paragraph are quoted:

```
>> In so doing,  support and maintenance costs can
```

```
>> be reduced, and users can benefit  from  having
>> ready  access  to others who are using the same
>> software.  Multi-user  licensing  is  also more
>> economical than  making  individual  purchases,
>> because there is no need for us to supply extra
>> disks, reference literature, etc. for all users
>> within the group.
```

Both the quoting style, and the quoting symbol used, can be configured on the Configure | Preferences | Editing 1 options page.

The Quote and Reformat command makes use of the Reformat command internally during its operation. As noted in the Reformat command, lines beginning with a period (.) are treated as blank lines in order to recognize text markup tags. As a result, the use of a quoting string that begins with a period will not produce the desired results, and should be avoided. All other symbols and characters are permissible.

4.6.11 Align Left

Menu: Paragraph > Align Left

Default Shortcut Key: Ctrl+F7

Macro function: AlignLeft()

The Align Left command moves the current line flush against the left edge, removing any indent which may have been present. The cursor is moved to the line below upon completion.

If text is selected, all lines within the selected range are affected.

This command will *not* cause words to be wrapped across lines. Use the Reformat command, with the desired Justification Style, for this purpose.

If the Align Left command is issued when a columnar selection is in force, the effect of the command will be to left align the selected text within the extent of the rectangular selection.

4.6.12 Align Center

Menu: Paragraph > Align Center

Default Shortcut Key: Ctrl+F8

Macro function: AlignCenter()

The Align Center command centers the current line within the current Text Width. The cursor is moved to the line below upon completion.

If text is selected, all lines within the selected range will be centered.

This command will *not* cause words to be wrapped across lines. Use the Reformat command, with the desired Justification Style, for this purpose.

If the Align Center command is issued when a columnar selection is in force, the effect of the command will be to center align the selected text within the extent of the rectangular selection.

4.6.13 Align Right

Menu: Paragraph > Align Right

Default Shortcut Key: Ctrl+F9

Macro function: AlignRight()

The Align Right command moves the current line flush against the right margin, as determined by the current Text Width. The cursor is moved to the line below upon completion.

If text is selected, all lines within the selected range are affected.

This command will *not* cause words to be wrapped across lines. Use the Reformat command, with the desired Justification Style, for this purpose.

If the Align Right command is issued when a columnar selection is in force, the effect of the command will be to right align the selected text within the extent of the rectangular selection.

4.6.14 Align Smooth

Menu: Paragraph > Align Smooth

Default Shortcut Key: Ctrl+F11

Macro function: AlignSmooth()

The Align Smooth command adjusts the current line to be flush against both the left and right margins. The right margin is determined according to the current Text Width. The cursor is moved to the line below upon completion.

Spaces are inserted alternately in the left, center and right portions of each line to minimize the appearance of *rivers and valleys* in the justified text.

If text is selected, all lines within the selected range are affected.

This command will *not* cause words to be wrapped across lines. Use the Reformat command, with the desired Justification Style, for this purpose.

If the Align Smooth command is issued when a columnar selection is in force, the effect of the command will be to smooth align the selected text within the extent of the rectangular selection.

4.7 Tools Menu

4.7.1 Macros

Menu: Tools > Macros

Default Shortcut Key: F8

Macro function: none

Boxer includes a powerful macro language than can be used to automate repetitive editing tasks, or to perform specialized processing on the text files you edit. Macros can be created in one of two ways: Macros can be recorded 'by example' by typing commands and/or insertable text within the macro dialog. When this is done, the macro code is written automatically, on-the-fly, in the editor window of the macro dialog. For more complex macros, the edit window can be used to write a macro by hand, or to make refinements to a macro that was recorded by example.

Boxer's macro language is similar in style to the C programming language, and will be quickly understood by anyone who has programmed in a high-level language, or in other macro/scripting languages. The macro dialog contains built-in lists of all the language's keywords, functions and operators, along with instant help information for each entry (see screen shots below). Boxer has been supplied with numerous example macros which are meant to illustrate the use of the language, as well as to provide genuinely useful services. For example, the `ExampleApplyHTML` macro will apply the necessary HTML declarations to make a simple text file into an HTML document.

Some people will want to dive right in, so here's a quick example:

Simple Macro Example, Step One

You've got a file that needs some repetitive editing. You need to delete the first four characters from the start of every third line. The file to be processed is open for editing, and the text cursor is sitting on the first line that needs adjustment. Here's how to create a macro to perform the editing required:

```
Issue the Tools|Macros command from the Main Menu
Click New
Press the Delete key four times
Press the Down Arrow key three times
Click Save
Enter a name for the macro
Click Run, as required
```

The resulting macro looks like this:

```
macro newmacro()
```

```
{
Delete;
Delete;
Delete;
Delete;
Down;
Down;
Down;
}
```

Simple Macro Example, Step Two

That's great, you say, but maybe your file is 300 lines long. Or 30,000 lines long. How can we make this macro work on the whole file?

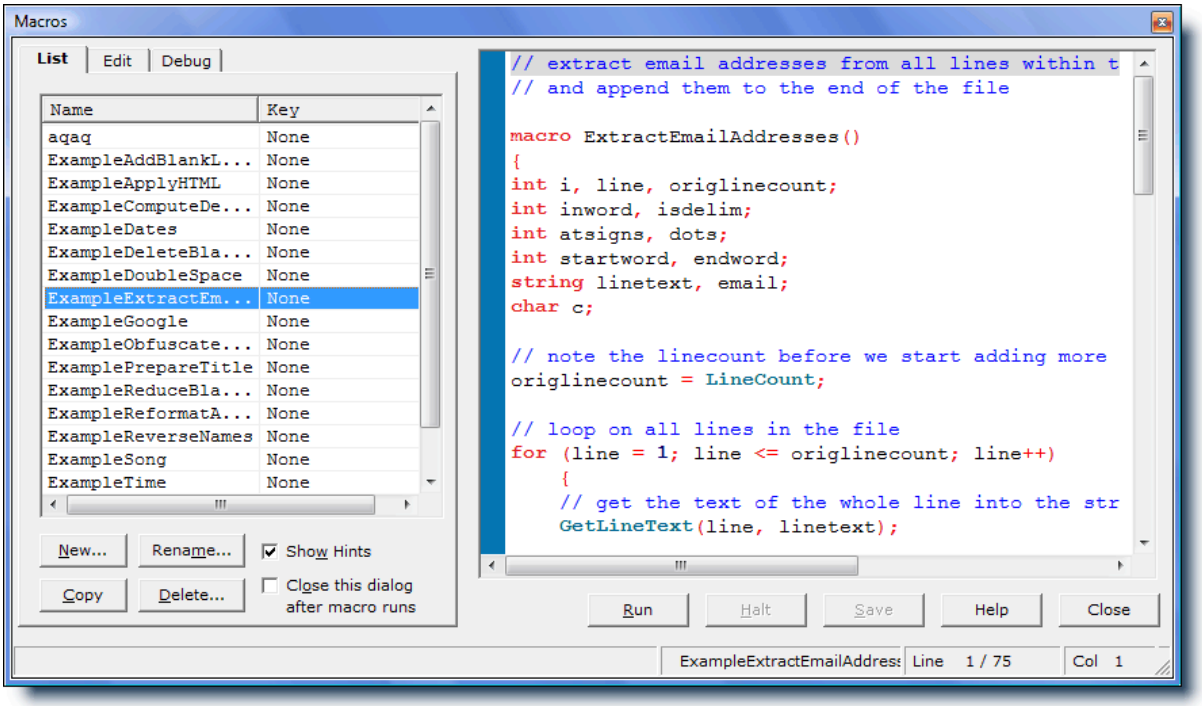
In Step One, the macro was written for you automatically, as you typed the editing commands. To handle a file of arbitrary length, we'll need to add a little code. Edit your macro to look like this:

```
macro newmacro()
{
int i;

for (i = 1; i <= LineCount(); i += 3)
    {
    Delete;
    Delete;
    Delete;
    Delete;
    Down;
    Down;
    Down;
    }
}
```

Click *Save*, and then *Run*. This macro loops through the file, counting by three, performing the necessary adjustments. Because it calls the function `LineCount()`, it will work for a file of any size.

See the following help topics for additional information about macros: Macro Language Reference, Macro Function Reference and Macro Examples.

The sections below cover the Macro Dialog in further detail...



List Tab

New

Use the *New* button to start a new macro. A new macro is created and control will switch to the *Edit Tab*. You will be able to name the macro later when you select the *Save* option.

Copy

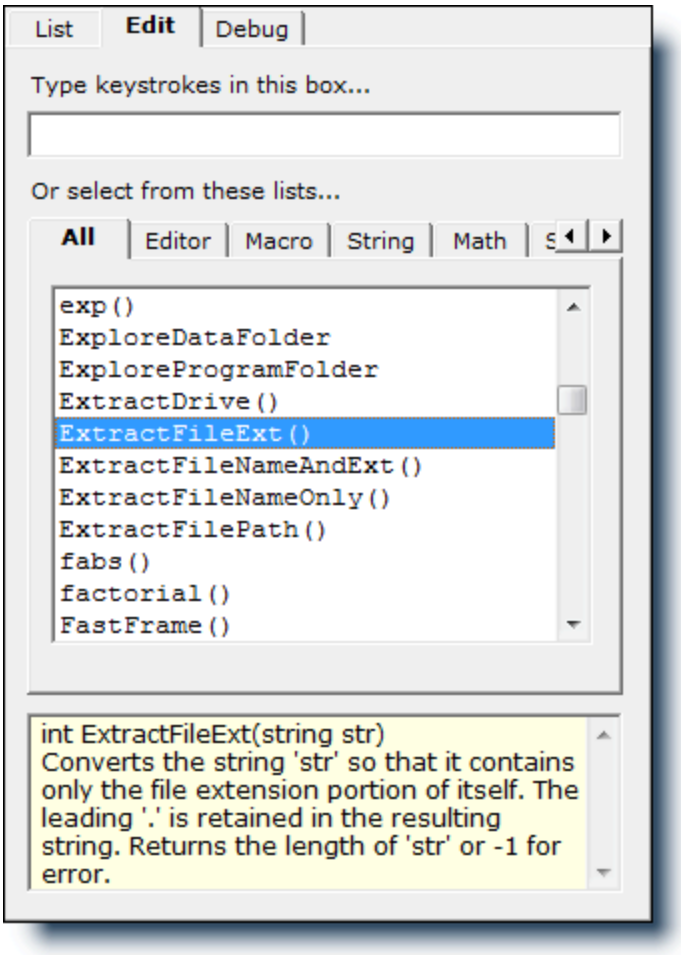
The *Copy* button will create a copy of the selected macro. You can then use the *Rename* button to rename the copy, if desired.

Rename

Use the *Rename* button to rename the selected macro.

Delete

Use the *Delete* button to delete the selected macro. A confirmation prompt will be supplied before the macro is deleted.



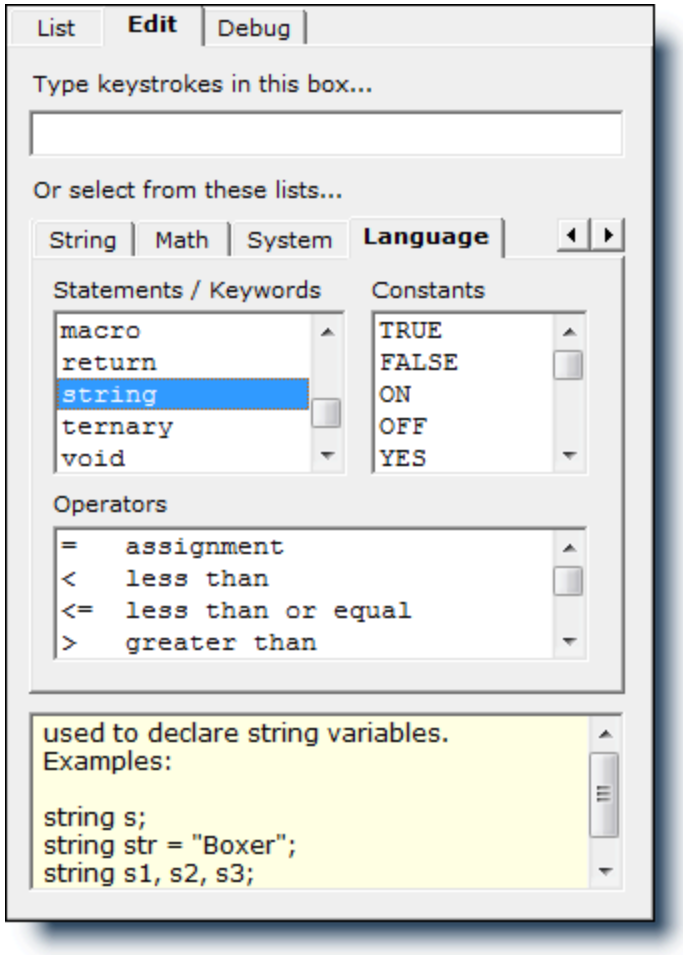
Edit Tab

The Edit Tab contains controls that can be used when composing a macro. If the macro is to be recorded 'by example', simply type the desired keys in the edit box at the top of the panel. You'll notice that the code of the macro is written automatically, as you type, in the editor window at the right. Feel free to switch to the edit window if changes are needed to the macro code. You can resume recording 'by example' at any time by positioning the text cursor in the editor window and returning focus to the edit box at the top left.

When composing a macro by hand, the lists on the *Edit Tab* will prove useful for recalling the macro language function names, keywords, and operators. Each time an entry is selected in a list, the help window at the bottom of the panel displays relevant information about the selected entry. You can insert the selected entry into the editor window by pressing *Enter* or by double-clicking.

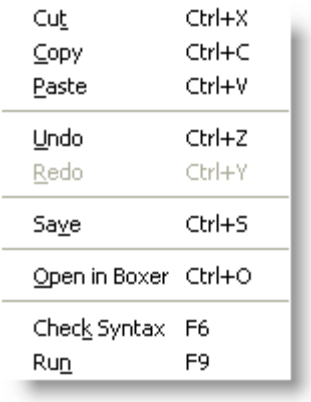
The *All* tab contains a list of all functions that are available in the macro language, regardless of their logical category. The *Editor*, *Macro*, *String* and *Math* tabs display function lists for each of those respective categories. *Editor* contains functions that map to commands available within the editor proper. *Macro* contains functions that are unique to the macro language. *String* contains functions that can be used to manipulate strings. *Math* contains functions that support mathematical operations.

The *Language Tab* contains lists of statements, keywords, constants and operators.

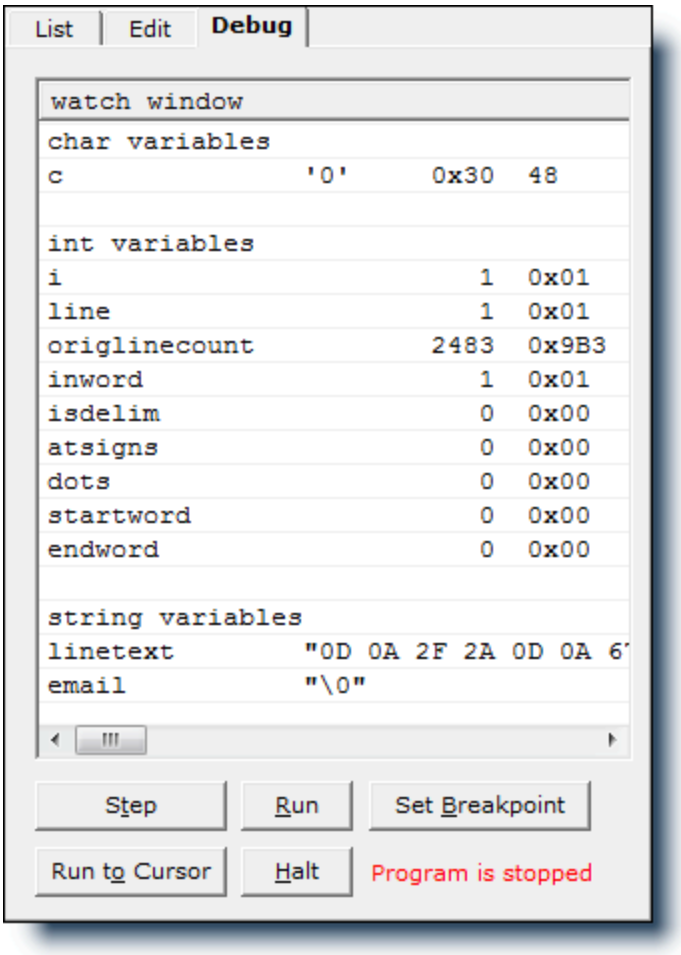


The editor window is used to edit the macro being composed. The macro is displayed with color syntax highlighting, just as if it were being edited in the editor proper. Although the macro editor window looks like a normal Boxer editing window, it is not. You will find that the standard editing and cursor movement commands are available within this window, but Boxer's advanced editing commands are not. If you are composing a complex macro, you might prefer to edit your macro within a normal editing window. For this reason, the editor offers an *Open in Boxer* command on its context menu.

The macro editor window has built-in help for the macro language. You can press *F1* when the cursor is sitting on a function name to view pop-up information for that function.



Additional editor functionality is available on the context menu by right clicking in the editor window when the *Edit Tab* is active.

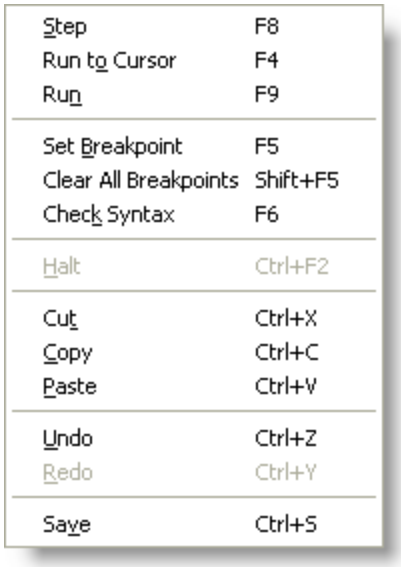


Debug Tab

The *Debug Tab* contains Boxer's integrated macro debugger. The debugger can be used

to control the execution of a macro and view a macro's variables as the macro is executed. The *Watch Window* shows a macro's variables, arranged by type, in both decimal and hexadecimal format. As the macro is executed, the *Watch Window* updates to show the current value of each variable. Note that there is no need to designate a variable as a watch variable; all variables are automatically added to the *Watch Window* each time the macro is debugged.

To begin debugging a macro, click the *Step* button. The *Step* button is used to execute a single line of code. You can click *Step* repeatedly to walk through the macro, line by line. To jump ahead in the macro, position the cursor in the editor window on the line of interest and click the *Run to Cursor* button. Execution will continue until the desired line is reached. The *Run to Cursor* button can be thought of as a one-time breakpoint. To ensure that execution will stop on a selected line every time, use the *Set Breakpoint* button. The line of interest will be highlighted in the editor window with a 'B'. To run the macro without single-stepping, click the *Run* button. *Run* causes the macro to run without interruption, until a breakpoint is hit. If no breakpoints are encountered, the macro will run to completion. Use the *Halt* button to terminate the execution of a macro.



Additional debugger functionality is available on the context menu by right clicking in the editor window when the *Debug* tab is active (see context menu above).

Assigning a Macro to a Key Sequence

There is theoretically no limit to the number of macros that can be created. All of the macros in Boxer's 'Macros' directory will be displayed in the macro list that appears on the *List* tab, and these macros can be run by clicking the nearby *Run* button. In addition, up to 50 macros will be displayed on the Tools | Run Macro submenu, and these macros can be executed directly from that menu. When more than 50 macros are present, those which sort lowest alphabetically will be the first to be omitted from the Run Macro submenu. If you want to force a certain macro to appear in the menu, you can do so by changing its filename to one that will rank higher in an alphabetic

sort.

You may wish to assign commonly used macros to a key assignment to make them easier to execute. There are 50 editor commands available for this use, named *Run Macro 1* to *Run Macro 50*. These commands appear in the command list on the Configure | Keyboard dialog. **In order to make a macro eligible for key assignment, its filename must end with a value from 1 to 50**. For example, if you name a macro `ProcessPayroll24.bm`, that macro can later be run by the key sequence that has been assigned to the *Run Macro 24* command.

Initially, the 50 *Run Macro N* commands are unassigned. Assigning a macro to a given key sequence is thus a two-step process:

1. Make sure that the filename of the macro ends with a value in the range 1-50, and does not conflict with other numbered macros.

2. Use the Configure | Keyboard dialog to assign a key sequence to the corresponding *Run Macro N* command.


If you rename a macro filename from `MyMacro4.bm` to `MyMacro12.bm`, the associated key assignment does not move automatically. The key assignment for the *Run Macro 4* command will always run whatever macro is numbered as 4. Therefore, you will need to visit the Configure | Keyboard dialog to make an adjustment after changing a macro's number.


A macro **cannot** be run from its assigned key sequence if that macro does not appear in the Tools | Run Macro submenu.

Running a Macro from the Command Line

A macro file can be run by naming it on the command line using the -M command line option flag. Please see the notes in that section for full details on this capability.

Running a Macro Automatically on Startup

There may be times when you want Boxer to perform a series of commands--or react to one or more configuration changes--every time the editor is launched. If a macro of the name `startup.bm` is found in the macros directory, it will be run automatically on startup.

Running a Series of Macros in Batch Mode

For some editing tasks it may be desirable to develop a series of macros to perform the necessary conversions. This approach may be desirable when the overall conversion is too complex to implement in a single macro, or when some steps of the conversion will need to be applied selectively on a case-by-case basis. If you have developed a set of macros, say `step1.bm`, `step2.bm` and `step3.bm`, these macros can be run in series from a *macro batch file* -- which might be named `do_it_all.bm --` and which names these files in succession:

```
step1.bm
```

```
step2.bm
step3.bm
```

Blank lines may appear within a *macro batch file*, but all other lines must contain the name of an existing macro file which is to be run.

For a clever tip that tells how to make use of your old Boxer/DOS, Boxer/TKO or Boxer/OS2 macros from within the Windows version of Boxer, see the tip near the bottom of the User Tools topic.

Storing Macro Variables From Run to Run

After a macro has completed, its variables are no longer available for study or use. Two macro functions can be used to store and recall macro variables so that they can be used again at a later time:

```
int WriteValue(string name, char/int/string/float val)
  Writes 'val' to the macro variable storage area named 'name'.
  'name' will be visible to other macros, so be careful to choose
  a unique identifier. Returns 1 for success or -1 for error.
  See also ReadValue(), EraseValue().

int ReadValue(string name, char/int/string/float val)
  Reads a value from the macro variable storage area named 'name'
  and places it into variable 'val'. The type of 'val' must agree
  with the type used when the value was written using WriteValue().
  Returns 1 for success or -1 for error.
  See also WriteValue(), EraseValue().
```

4.7.2 Macro Language Reference

Data Types

Boxer's Macro Language supports the following data types:

```
string
char
int
float
```

A `string` can hold a series of characters up to 2,048 bytes in length. The end of a string is marked with a Null character (ASCII 0). A string constant is enclosed within double quotes.

The `char` data type is an 8-bit, unsigned data type which can hold values in the range 0 to 255. A character constant is enclosed within single quotes.

The `int` data type is a 32-bit, signed data type which can hold integer values in the range -2,147,483,648 to 2,147,483,647.

The `float` data type is a double precision, signed data type that can hold values in the range 2.2250738585072014e-308 to 1.7976931348623158e+308.

Keywords

The following words are reserved keywords and may not be used as variable names:

```
break       int         true
continue    char        false
do          string      yes
else        float       no
for         void        on
goto                    off
if
macro
return
while
```

The keywords listed above are case sensitive, and must be entered in lowercase. The symbolic constants in the third column (`true, false, yes, no, on, off`) are an exception: they can appear in lowercase, uppercase, or even in mixed case.

Arithmetic Operators

The following arithmetic operators are supported:

Operator	Meaning
+	addition
-	subtraction
*	multiplication
/	division
%	modulus
++	increment
--	decrement

The modulus operator (`%`) returns the remainder from an integer division operation. For example, the expression `n = 7 % 4` will result in `n` receiving the value `3`, since `7 / 4` leaves a remainder of `3`.

The increment and decrement operators can be used to increase or decrease an integer variable by `1`. The expression:

```
i++;
```

is equivalent to:

```
i = i + 1;
```

The ++ and -- operators can be used in either prefix or postfix location. If i has an initial value of 3, the statement:

```
n = i++;
```

will leave n with the value of 3, while i is incremented to 4. The incrementing of i occurs *after* the assignment due to the postfix location.

Assuming i again starts with a value of 3, the statement:

```
n = --i;
```

will leave n with a value of 2 and i with a value of 2. The decrementing of i occurs *before* the assignment due to the prefix location.

The addition (+) operator has been overloaded to support string concatenation. The following statements:

```
string s1 = "Boxer ";
string s2 = "Text Editor";
string s3 = s1 + s2;
```

would result in s3 having the value: "Boxer Text Editor"

Assignment Operators

The following assignment operators are supported:

Operator	Meaning
=	assignment
+=	addition assignment
-=	subtraction assignment
*=	multiplication assignment
/=	division assignment
%=	modulus assignment
&=	bitwise AND assignment
|=	bitwise OR assignment
^=	bitwise XOR assignment
<<=	left shift assignment
>>=	right shift assignment

The assignment operator (=) should be familiar to all. The other operators which each conclude with = all represent a shorthand notation. For example, the statement:

```
i += 5;
```

is equivalent to:

```
i = i + 5;
```

The += operator has been *overloaded* to support string concatenation. The following statements:

```
string str = "Boxer ";
str += "Text Editor";
```

would result in `str` having the value: `"Boxer Text Editor"`

The last five operators listed above are bitwise assignment operators. Their function is analogous to the += operator; see the Bitwise Operators section of this topic for some additional detail.

Boolean Operators

The following Boolean operators are supported:

Operator	Meaning
==	equal
!=	not equal
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
&&	logical AND
||	logical OR
!	logical NOT (unary negation)
~=	case insensitive string comparison

The operators ==, !=, <, >, <= and >= have been overloaded to allow operations on strings. A string is considered greater than another string if it would appear higher in an alphabetic sort. In other words, the statement:

```
if ("apple" < "zebra")
```

evaluates to TRUE.

The first nine operators above are standard to most high-level languages. The last operator is specific to Boxer's Macro Language, and permits strings to be compared

without case sensitivity. For example, the statement:

```
if ("MasterCard" ~= "mastercard")
```

would evaluate to TRUE.

Bitwise Operators

The following bitwise operators are supported:

Operator	Meaning
&	bitwise AND
|	bitwise OR
^	bitwise XOR
<<	left shift
>>	right shift
~	one's complement (unary)

A full discussion of bitwise arithmetic would be beyond the scope of this language reference. For those who are interested, any introductory book on the C programming language would be a suitable reference. The information below will be sufficient to remind those with prior experience of the function of each operator:

& Sets a bit to 1 in the result if and only if both of the corresponding bits in its operands are 1, and to 0 if the bits differ or both are 0. Example: `9 & 1` yields `1`.

| Sets a bit to 1 in the result if one or both of the corresponding bits in its operands are 1, and to 0 if both of the corresponding bits are 0. Example: `9 | 2` yields `11`.

^ Sets a bit in the result to 1 when the corresponding bits in its operands are different, and to 0 when they are the same. Example: `7 ^ 4` yields `3`;

<< Shifts the first operand the number of bits to the left specified in the second operand, filling with zeros from the right. Example: `2 << 3` yields `16`.

>> Shifts the first operand the number of bits to the right specified in the second operand, discarding the bits that 'fall off' at the right. Example: `34 >> 2` yields `8`.

~ Inverts each bit in the operand, changing all ones to zeros and all zeros to ones. Example: `~0xFFFF0000` yields `0x0000FFFF`.


The large majority of users will never find a need for bitwise arithmetic, but it has been included in the interest of completeness.

Operator Precedence

The following table summarizes operator precedence and order of evaluation for the various operators supported by Boxer's Macro Language. Operators with the strongest/highest precedence are listed first:

Operator	Evaluates
( ) [ ]	left to right
! ~ ++ -- -	right to left
* / %	left to right
+ -	left to right
<< >>	left to right
< <= > >=	left to right
== != ~=	left to right
|	left to right
&	left to right
^	left to right
&&	left to right
||	left to right
? :	right to left
= += -= etc.	right to left
,	left to right

Parentheses can be used when required to ensure that the order of evaluation occurs as desired. For example:

```
n1 = 3 * 5 + 4;
```

assigns `19` to `n1`, while:

```
n1 = 3 * (5 + 4);
```

assigns `27` to `n1`.

Because the assignment operator (`=`) is evaluated from right to left, a construction such as the following is possible:

```
int i, j, k;
i = j = k = 0;
```

`k` is assigned the value `0`, `j` is assigned the value of `k`, and `i` is assigned the value of `j`.

Character Constants

Boxer's Macro Language recognizes the standard character constants which have been popularized by the C programming language:

Sequence	Meaning	Decimal Value
'\b'	Backspace	8
'\f'	Formfeed	12
'\n'	Newline	10
'\r'	Carriage Return	13
'\t'	Tab	9
'\\'	Backslash	92
'\''	Single Quote	39
'\"'	Double Quote	34
'\0'	Null	0

In addition, Boxer will recognize a backslash (`\`) followed by three *octal* digits as the character whose ASCII value is given by the digits used. For example, `'\101'` could be used to represent a capital `A`, since its ASCII value, in octal, is 101.

Character constants can be used in any place that a `char` data type is expected, or within a double-quoted string: `"this is a string with a newline at the end.\n"`

Numeric Constants

Numeric `int` constants can be specified in either *decimal* or *hexadecimal* format:

```
int n1 = 32;
int n2 = 0x20;
```

Each of these assignments supplies the value `32` to `n1` or `n2`.

Numeric `float` constants can be specified in any of the following forms:

```
float x1 = 500;
float x2 = 500.0;
float x3 = 5e2;
float x4 = 5e02;
float x5 = 5.0e2;
float x6 = 5.0e02;
float x7 = 5.0e+2;
float x7 = 5.0e+02;
```

Each of these assignments results in the value `500` being assigned to the variable being declared.

For floating point values less than 1, the minus sign can be used to designate exponentiation. All of the following examples represent the number .05:

```
.05
0.05
5e-2
5e-02
5.0e-2
5.0e-02
```

Symbolic Constants

The following symbolic constants are recognized:

Name	Value
TRUE	1
FALSE	0
YES	1
NO	0
ON	1
OFF	0

These constants can be used in place of the values 0 and 1 to make a macro more readable. For example, you can write:

```
ViewBookmarks(ON);
```

instead of:

```
ViewBookmarks(1);
```

Declaring Variables

Variable names can be up to 32 characters in length and must not conflict with the names of any keywords or internal functions. Variable names can use alphanumeric characters and the underscore (_), but they must not start with a digit. All variables must be declared before use. Initialization of variables can be done at declaration-time, but this is not required. Uninitialized variables will be zero-filled automatically.

Boxer's Macro Language supports a flexible syntax for declaring variables. All of the following examples are legal declarations when they appear at the top of a macro, before other executable statements:

```
string s1;
string s2 = "Boxer";
string s3, s4, s5;
string s6 = "abc", s7, s8 = "def";
```

```
char c1;
char c2 = 'A';
char c3, c4, c5;
char c6, c7 = 'x', c8;

int n1;
int n2 = 10;
int n3, n4, n5;
int n6, n7 = -4, n8;

float x1;
float x2 = 1.05;
float x3 = 1.2e04;
float x4, x5, x6;
float x7, x8 = 7.75, x9;
```

In the spirit of the C programming language, Boxer's macro language also allows a `string` variable to be declared as an array of characters. The declaration:

```
char str[100];
```

is (for most purposes) functionally equivalent to the declaration:

```
string str;
```

for declaring a variable which can hold a short string of characters. See the *String Subscripting* section below for details on when the former style might be required.

Conditional Statements

Boxer's Macro Language supports three different conditional statements: `if`, `if-else` and the ternary statement. An `if` statement will be executed if the expression in parentheses evaluates to a non-zero result. Below are examples of the three conditional statements:

```
if (LineCount() > 10000)
   {
   longfile = true;
   }


if (LineCount() > 10000)
   {
   longfile = true;
   }
else
   {
   longfile = false;
   }


longfile = (LineCount() > 10000) ? true : false;
```

In the first example, the variable `longfile` is set `TRUE` if the return from the function `LineCount()` is greater than `10000`. In the second example, an `if-else` statement is used to additionally set `longfile` to `FALSE` if the condition is *not* met.

The final example illustrates the ternary statement, and its effect is identical to the `if-else` example immediately above it. If the condition within parentheses evaluates to `TRUE`, the expression immediately following the `?` is evaluated. If not, the expression after the `:` is evaluated. A ternary statement is effectively a compact `if-else` statement.


The ternary statement in Boxer's Macro Language is modeled after that of the C programming language, with one exception. In Boxer macros, the parentheses around the conditional expression are *required*, in C these parentheses are optional.


When a single statement is conditional upon an `if` or `if-else` statement, as is shown in the examples above, the use of curly braces `{ }` is not required. Curly braces *are* required when two or more statements are to be conditionally executed, or when those statements are the subject of a looping statement.

Looping Statements

Boxer's Macro Language supports three different looping statements: `for`, `while` and `do-while`. A loop statement will continue looping so long as the 'test' expression in parentheses evaluates to a non-zero result. Below are examples of each of these statements:

```
// find the longest line in the file
for (line = 1, longest = 0; line <= LineCount(); line++)
   if ((n = LineLength(line)) > longest)
        longest = n;



// find the longest line in the file
line = 1;
longest = 0;
while (line <= LineCount())
   {
   if ((n = LineLength(line)) > longest)
        longest = n;

   line++;
   }



// find the longest line in the file
line = 1;
longest = 0;
do
```

```
    {
    if ((n = LineLength(line)) > longest)
         longest = n;

    line++;
    }
  while (line < LineCount());
```

The three loops above are functionally equivalent to one another, with one exception that will be discussed below.

The `for` loop is the most compact, since it permits the three elements of a loop's control to be specified on a single line: the initialization, the test, and the increment. These are found within the parentheses of the `for` loop and are separated by semi-colons. When a `for` loop is first executed, the initialization section is performed, and the test section is evaluated. If the test evaluates to a non-zero result, the statement(s) in the body of the loop are processed. At the end of the loop, the increment section is processed. Control then passes again to to the test section, to the body, and so on.


Boxer's Macro Language supports a very flexible `for` loop structure. The initialization, test and increment sections are each optional. Moreover, multiple initializations can be performed by separating the statements with the comma operator.

The `while` loop is a simpler loop, in that the only required control element that must be supplied is the test. For illustration purposes, the `while` loop above was written to be identical in function to the `for` loop above it. In fact, every `for` loop can be written as a `while` loop, and every `while` loop can be written as a `for` loop. A `for` loop is typically used when one needs to initialize and increment a loop index. A `while` loop is typically used when a single condition is sufficient to control the flow of the loop.

A `do-while` loop is essentially an upside-down `while` loop. A `do-while` loop tests at the bottom, whereas a `while` loop tests at the top. A `do-while` loop should be used in those cases where the loop is always to be be executed at least once. That leads us to why the `do-while` example above is not exactly equivalent to the `for` and `while` loops above it. If the current file is empty, the `for` and `while` loops above will not be executed. The `LineCount()` function will return `0` and the initial test will fail. In the `do-while` loop, the `LineCount()` call isn't made until the bottom of the loop. In the case of an empty file, the body of the loop would be processed and the `LineLength()` call would fail because the `line` parameter would be out of range.


Sometimes the need arises to construct a 'forever' loop; one which will run until some condition within the body of the loop is satisfied and a `break` statement is executed. Both the `for` and `while` loops can be used for this purpose. Here are two examples:

```
// loop until the user enters the right answer
```

```
for (;;)
   {
   GetString("What's the capital of Arizona?", answer);

   if (answer ~= "Phoenix")
        break;
   }

// loop until the user enters the right answer
while (TRUE)
   {
   GetString("What's the capital of New Hampshire?", answer);

   if (answer ~= "Concord")
        break;
   }
```


Notice that these examples used the `~=` operator to ensure that the user's response was not rejected due to improper case.

Alert readers might notice that the above examples could be more neatly implemented using a `do-while` loop, since this is a case where the loop always wants to be run once, and the test can be more logically placed at the bottom of the loop:

```
do
   GetString("What's the capital of California?", answer);
while (strcmpi(answer, "Sacramento") != 0);
```


This example uses the `strcmpi()` function to perform a case insensitive string comparison, because the `~=` operator does not have a companion *string-does-not-match* operator.

The break Statement

The `break` statement can be used to exit from a loop prematurely. Control passes to the next statement following the loop which has been exited. For example:

```
// loop on all lines in the file
for (i = 1; i <= LineCount(); i++)
   {
   // exit the loop if a line is longer than 1000 characters
   if (LineLength(i) > 1000)
        break;
   }

// control passes to here after break
New;
```

The continue Statement

The `continue` statement can be used to jump to the bottom of a loop prematurely. Control passes to an imaginary label at the end of the loop. For example:

```
// loop on all lines in the file
for (i = 1; i <= LineCount(); i++)
   {
   // exit the loop if a line is longer than 1000 characters
   if (LineLength(i) > 1000)
        continue;

   // ... other processing ...

   // continue jumps to here
   }
```

The goto Statement

The `goto` statement can be used to jump unconditionally to a label. Control passes to the next statement after the label. For example:

```
// loop on all lines in the file
for (i = 1; i <= LineCount(); i++)
   {
   // exit the loop if a line is longer than 1000 characters
   if (LineLength(i) > 1000)
        goto toolong;

   // ... other processing ...

   toolong:
   // goto jumps to here

   // ... other processing ...
   }
```

The return Statement

The `return` statement can be used to end a macro prematurely. If a return statement is not encountered, a macro will run until the closing curly brace in the body of the macro is encountered.

Function Calls

Boxer's Macro Language includes a wide variety of functions that provide access to the editor's commands, configuration settings, and to string and math libraries. The function set is documented in the Macro Function Reference, as well as in the Macro Dialog itself.

When making a function call, care should be taken to ensure that the parameters supplied to the function match the declared type(s) that the function expects to receive. Boxer is able to trap missing and/or mismatched parameters in most cases, but unexpected results can occur when invalid parameters are supplied.

Function names are not case sensitive; Boxer will accept function names that do not

match the function name with regard to character case.

If a function does not require parameters, it is not necessary to supply parentheses at the end of the function name. For example:

```
LineCount();
```

and

```
LineCount;
```

are functionally equivalent, because the `LineCount` function does not require any parameters. That said, the practice of using `()` on all function calls can help to distinguish function names from variable names.

Simple expressions can be supplied to in a function call without difficulty, and they will be evaluated as expected before being sent to the function for processing. For example:

```
max(3 * 45, 4 * 90);
```

is a legitimate construction that might be used in calling the `max()` function. If you find that you are getting unexpected results in a case like this, introduce a temporary variable to hold the value of the expression, and then supply the variable to the function in place of the expression.

String Subscripting

Arrays are not supported in the classical sense; it's not possible to declare an array of `int` or `float` variables, for example. But Boxer's Macro Language does recognize a `string` variable to be an array of elements of type `char`, and allows those elements to be accessed individually through the use of subscripts. The first character within a string is located at index `0`, the second character is at index `1`, etc. In the following example:

```
string str = "BOXER";
char c1;
c1 = str[2];
```

the character variable `c1` would be assigned the value `'X'`.

Likewise, a `string` variable can be modified by assigning individual elements within the string using subscripting:

```
string s1 = "water";
s1[0] = 'w';
s1[1] = 'i';
s1[2] = 'n';
s1[3] = 'e';
s1[4] = '\0';
```

This code fragment has the effect of changing the content of string variable `s1` from `"water"` to `"wine"`. Notice that the null character (`'\0'`) was used to shorten the string from five characters to four.

String subscripting makes it possible to use a string variable in the way that an array might be used. Here's an example that totals the number of occurrences of each letter within an input string:

```
macro array_example()
{
int i;
string input = "now is the time for all good men to come to the
aid of their country.";
char tally[256];        // note that all elements are initially
set to zero

// loop to process all characters in the input string
for (i = 0; input[i] != '\0'; i++)
   tally[input[i]]++;

// open a new, untitled file
New;

// report the results for lowercase letters
for (i = 'a'; i <= 'z'; i++)
   printf("letter %c occurred %d time(s)\n", i, tally[i]);
}
```

Had the `tally` array been declared as a `string` type, Boxer's built-in range checking would have prevented the string from being used in the way that was shown above. By declaring the string as a character array of sufficient size, the macro processor is forewarned that the code may later index into the string beyond the terminating null character.

Due to the capacity of the `char` data type (0-255), the utility of the above technique is limited to applications in which the maximum number of occurrences would be less than 256.

Type Conversions

Boxer's Macro Language will automatically convert between data types whenever possible in order to resolve an expression that involves mismatched data types. Here are some examples:

```
string s1 = 'A';        // result: s1 gets "A" (char to string)
string s2 = 65;         // result: s2 gets "A" (int to string)
string s3 = 65.0;       // result: s3 gets "A" (float to string)

char c1 = 65;           // result: c1 gets 'A' (int to char)
char c2 = "A";          // result: c2 gets 'A' (string to char)
char c3 = 65.0;         // result: c3 gets 'A' (float to char)
```

```
   int n1 = 'A';           // result: n1 gets 65; (char to int)
   int n2 = "123";         // result: n2 gets 123 (string to int)
   int n3 = 123.45;        // result: n3 gets 123 (float to int)

   float x1 = 'A'          // result: x1 gets 65.0 (char to float)
   float x2 = 65;          // result: x2 gets 65.0 (int to float)
   float x3 = "123.45";    // result: x3 gets 123.45 (string to
float)
```

Comments

Comments can be placed throughout a macro to help document the code. Two types of comments are supported, block comments and end-of-line comments:

```
/* this is a multi-line
   block comment */

int n1 = 7;       // this is an end-of-line comment
```

4.7.3 Macro Function Reference

Function	Prototype and Description
abs()	int abs(int n) Returns the absolute value of 'n'.
acos()	float acos(float x) Returns the arc cosine of 'x' in radians. 'x' must be in the range -1 to 1.
ActiveClipboard	int ActiveClipboard Returns the number of the active clipboard. The Windows Clipboard is number 0; private clipboards are numbered 1 to 8.
ActiveSpellChecking()	int ActiveSpellCheck(int mode) Enables or disables Active Spell Checking according to 'mode'.
AlignCenter	Issues the Align Center command
AlignLeft	Issues the Align Left command
AlignRight	Issues the Align Right command
AlignSmooth	Issues the Align Smooth command
ANSIChart	Issues the ANSI Chart command
ANSItoOEM	Issues the ANSI to OEM command
Append	Append Append the selected text to the current clipboard. If no text is selected, the current line is appended to

	the clipboard.
AppendToClipboard()	int AppendToClipboard(string str, int n) Appends string 'str' to Clipboard 'n'. Returns the total length of the text on the clipboard, or -1 for error. The Windows Clipboard is number 0; private clipboards are numbered 1 to 8. See also PutClipboardText().
ApplyHighlighting	Issues the Apply Highlighting command
ArrangeIcons	Issues the Arrange Icons command
ASCIItoEBCDIC	Issues the ASCII to EBCDIC command
asin()	float asin(float x) Returns the arc sine of 'x' in radians. 'x' must be in the range -1 to 1.
atan()	float atan(float x) Returns the arc tangent of 'x' in radians.
atof()	float atof(string str) Returns the floating point value of the number described by string 'str'.
atoi()	int atoi(string str) Returns the integer value of the decimal number described by string 'str'.
AutoNumber	Issues the Auto-Number command
Backspace	Issues the Backspace command
Backtab	Issues the Backtab command
Beep()	Beep(int freq, int duration) Makes a sound through the PC speaker using the supplied values for frequency and duration. Frequency is in Hz and duration is in milliseconds. Beep(1000, 300) produces a standard beep.
BookmarkManager	Issues the Bookmark Manager command
BottomOfPage	Issues the Bottom of Page command
BringUserListsToTop	Issues the Bring User Lists to Top command
BrowseForFilename()	BrowseForFilename(string fn, int mustexist) Browse for a filename using a standard Windows open dialog and place the selected filename in 'fn'. If 'mustexist' is non-zero, the selected filename must already exist. If 'mustexist' is 0, a new filename can be selected. Returns 1 for success or -1 for error.
ByteCount	int ByteCount Returns the number of characters in the current file.

Calculator	Issues the Calculator command
Calendar	Issues the Calendar command
Cascade	Issues the Cascade command
CascadeHorizontal	Issues the Cascade Horizontal command
CascadeVertical	Issues the Cascade Vertical command
CaseInvert	Issues the Case Invert command
CaseLower	Issues the Case Lower command
CaseSentences	Issues the Case Sentences command
CaseTitle	Issues the Case Title command
CaseUpper	Issues the Case Upper command
CaseWords	Issues the Case Words command
ceil()	float ceil(float x) Returns (as a float) the smallest integer not less than 'x'. Example: ceil(1.5) returns 2.0.
ChangeString()	int ChangeString(string str1, str2, str3) Searches 'str1' and changes all occurrences of 'str2' to the string 'str3'. Returns the number of changes made or -1 for error. The search is case sensitive. Regular expressions are not recognized. If 'str3' is an empty string, the effect will be to delete all occurrences of 'str2' within 'str1'.
ChangeStringi()	int ChangeStringi(string str1, str2, str3) Searches 'str1' and changes all occurrences of 'str2' to the string 'str3'. Returns the number of changes made or -1 for error. The search is case insensitive. Regular expressions are not recognized. If 'str3' is an empty string, the effect will be to delete all occurrences of 'str2' within 'str1'.
ChangeStringRE()	int ChangeStringRE(string str1, str2, str3) Searches 'str1' and changes all instances matching 'str2' to the string 'str3'. Returns the number of changes made or -1 for error. The search is case sensitive. Regular expressions ARE recognized in 'str2'. If 'str3' is an empty string, the effect will be to delete all occurrences of 'str2' within 'str1'.
ChangeStringREi()	int ChangeStringREi(string str1, str2, str3) Searches 'str1' and changes all instances matching 'str2' to the string 'str3'. Returns the number of changes made or -1 for error. The search is case insensitive. Regular expressions ARE recognized in 'str2'. If 'str3' is an empty string, the effect will be to delete all occurrences of 'str2' within 'str1'.

CheckWord	Issues the Check Word command
ClearAllBookmarks	Issues the Clear All Bookmarks command
ClearAllClipboards	Issues the Clear All Clipboards command
ClearClipboard()	ClearClipboard(int n) Clears the content of Clipboard 'n'. The Windows Clipboard is number 0; private clipboards are numbered 1 to 8.
ClearClosedTabsList	Issues the Clear Closed Tabs List
ClearRecentFilesList	Issues the Clear Recent Files List command
ClearRecentProjectsList	Issues the Clear Recent Projects List command
ClearUndo	Issues the Clear Undo command
Close	Issues the Close command
CloseAll	Issues the Close All command
CloseAllButActive	Issues the Close All But Active command
ColorChart	Issues the HTML Color Chart command
Column	int Column Returns the column number of the text cursor in the current file, or -1 for error. The column returned is 1-based, not 0-based, and does not give consideration to the display value of any tabs that may appear in the line. See also DisplayColumn().
Comment	Issues the Comment command
ConfigureColors	Issues the Configure Colors command
ConfigureCtagsFunctionIndexing	Issues the Configure Ctags Function Indexing command
ConfigureKeyboard	Issues the Configure Keyboard command
ConfigurePreferences	Issues the Configure Preferences command
ConfigurePrinterFont	Issues the Configure Printer Font command
ConfigureScreenFont	Issues the Configure Screen Font command
ConfigureSyntaxHighlighting	Issues the Configure Syntax Highlighting command
ConfigureTemplates	Issues the Configure Templates command
ConfigureTextHighlighting	Issues the Configure Text Highlighting command
ConfigureToolbar	Issues the Configure Toolbar command
ConfigureUserTools	Issues the Configure User Tools command

Copy	Copy Copy the selected text to the current clipboard. If text is not selected, the current line is copied to the clipboard.
CopyFile()	int CopyFile(string oldname, string newname) Copies the file 'oldname' to the file 'newname', overwriting the output file if it already exists. Returns 1 for success or -1 for error.
CopyFilename	Issues the Copy Filename command
cos()	float cos(float x) Returns the cosine of 'x'. The angle 'x' must be in radians.
cosh()	float cosh(float x) Returns the hyperbolic cosine of 'x'. The angle 'x' must be in radians.
CreateDirectory()	int CreateDirectory(string dir) Creates a new directory according to the fully qualified filepath in 'dir'. Returns 1 for success or -1 for error.
CtagsFunctionIndex	Issues the Ctags Function Index command
Cut	Cut Cut the selected text to the current clipboard. If text is not selected, the current line is cut to the clipboard.
CutAppend	CutAppend Cut the selected text and append it to the current clipboard. If text is not selected, the current line is cut and appended to the clipboard.
Declaration	Issues the Declaration command
Decrement()	int Decrement(int n) Subtracts 'n' from the value at the text cursor and places the result in the text file. Returns the result of the operation or -1 for error. If 'n' is not supplied the Decrement dialog will appear when the macro is run.
Delete	Delete Deletes the character at the text cursor, or the selected text.
DeleteBlankLines	Issues the Delete Blank Lines command
DeleteBookmarkedLines	Issues the Delete Bookmarked Lines command
DeleteDuplicateLines	Issues the Delete Duplicate Lines command
DeleteFile()	int DeleteFile(string name) Deletes the fully qualified filepath 'name' from the

	disk, without requesting confirmation. Returns 1 for success or -1 for error.
DeleteLine	int DeleteLine(int n) Deletes line 'n' in the current file. Returns 1 for success or -1 for error. If 'n' is not supplied, the current line is deleted.
DeleteLinesThatBeginWith	int DeleteLinesThatBeginWith(string str) Delete lines that begin with the string 'str'. Returns 1 for success or -1 for error. If 'str' is not supplied a dialog will appear when the macro is run.
DeleteLinesThatContain	int DeleteLinesThatContain(string str) Delete lines that contain the string 'str'. Returns 1 for success or -1 for error. If 'str' is not supplied a dialog will appear when the macro is run.
DeleteLinesThatDoNotBeginWith	int DeleteLinesThatDoNotBeginWith(string str) Delete lines that do not begin with the string 'str'. Returns 1 for success or -1 for error. If 'str' is not supplied a dialog will appear when the macro is run.
DeleteLinesThatDoNotContain	int DeleteLinesThatDoNotContain(string str) Delete lines that do not contain the string 'str'. Returns 1 for success or -1 for error. If 'str' is not supplied a dialog will appear when the macro is run.
DeleteLinesThatDoNotEndWith	int DeleteLinesThatDoNotEndWith(string str) Delete lines that do not end with the string 'str'. Returns 1 for success or -1 for error. If 'str' is not supplied a dialog will appear when the macro is run.
DeleteLinesThatEndWith	int DeleteLinesThatEndWith(string str) Delete lines that end with the string 'str'. Returns 1 for success or -1 for error. If 'str' is not supplied a dialog will appear when the macro is run.
DeleteNextWord	Issues the Delete Next Word command
DeletePreviousWord	Issues the Delete Previous Word command
DeleteToEndOfLine	Issues the Delete to End of Line command
DeleteToStartOfLine	Issues the Delete to Start of Line command
Deselect()	Deselect(int mode) Releases the text selection, if one exists. If 'mode' is 0, the text cursor is placed at the beginning of the selection. If 'mode' is 1, the text cursor is placed at the end of the selection. If 'mode' is 2, the current position of the text cursor is maintained. Returns 1 for success or -1 for error.
DisplayColumn	int DisplayColumn Returns the column number of the text cursor in the current file, or -1 for error. The column returned is

	1-based, not 0-based, and gives consideration to the display value of any tabs that may appear in the line. See also Column().
Divide()	int Multiply(int n) Divides the value at the text cursor by 'n' and places the result in the text file. Returns the result of the operation or -1 for error. If 'n' is not supplied the Divide dialog will appear when the macro is run.
Down	int Down(int n) Issues the Down command 'n' times. Returns the number of commands performed or -1 for error. The argument 'n' is optional; if it is not provided a single command is performed.
DuplicateAndIncrement	Issues the Duplicate and Increment command
DuplicateLine	Issues the Duplicate Line command
e	float e Returns the value of Euler's number 'e', which is approximately 2.7182818285.
EBCDICtoASCII	Issues the EBCDIC to ASCII command
EditClipboard()	EditClipboard(int n) Opens Clipboard 'n' in a window for editing. The Windows Clipboard is number 0; private clipboards are numbered 1 to 8.
EndOfFile	Issues the End of File command
EndOfLine	Issues the End of Line command
Enter	Issues the Enter command
EraseValue()	int EraseValue(string name) Erases the variable 'name' from the macro variable storage area. Returns 1 for success or -1 for error. See also ReadValue(), WriteValue(), ValueExists().
ErrorChart	Issues the Error Chart command
Exit	Exit Issues the File|Exit command to close the editor. If one or more files have not been saved a prompt will appear when the macro is run.
exp()	float exp(float x) Returns the value 'e' raised to the 'x'.
ExploreDataFolder	Issues the Explore Data Folder command
ExploreProgramFolder	Issues the Explore Program Folder command

ExtractDrive()	int ExtractDrive(string str) Converts the string 'str' so that it contains only the drive designation portion of itself (eg 'C:'). Returns the length of 'str' or -1 for error.
ExtractFileExt()	int ExtractFileExt(string str) Converts the string 'str' so that it contains only the file extension portion of itself. The leading '.' is retained in the resulting string. Returns the length of 'str' or -1 for error.
ExtractFileNameAndExt()	int ExtractFileNameAndExt(string str) Converts the string 'str' so that it contains the filename.ext portion of itself. Returns the length of 'str' or -1 for error.
ExtractFileNameOnly()	int ExtractFileNameOnly(string str) Converts the string 'str' so that it contains only the filename portion of itself. Returns the length of 'str' or -1 for error.
ExtractFilePath()	int ExtractFilePath(string str) Converts the string 'str' so that it contains only the filepath portion of itself. The trailing backslash is retained in the resulting string. Returns the length of 'str' or -1 for error.
fabs()	float fabs(float x) Returns the absolute value of 'x'.
factorial()	int factorial(int x) Returns the value of x factorial, also known as x! Returns -1 for error or overflow.
FastFrame()	int FastFrame(int style) Surrounds the columnar selection with a frame according to 'style'. When 'style' is in the range 1 to 11, a corresponding line style from the Fast Frame dialog is used. If 'style' is not supplied the Fast Frame dialog will appear when the macro runs. Returns 1 for success or -1 for error.
FileCount	int FileCount Returns the number of files currently open in the editor.
FileExists()	int FileExists(string filepath) Returns 1 if 'filepath' exists, 0 if it does not exists, or -1 for error.
FileName	int FileName(string fn) Fills 'fn' with the full path of the current file. Returns the length of the filepath or -1 for error.
FilePicker	Issues the File Picker command
FileProperties	Issues the File Properties command

FileTabsBottom	Issues the File Tabs Bottom command
FileTabsTop	Issues the File Tabs Top command
FillWithString()	int FillWithString(string str) Fills the selected region with 'str'. Returns 1 for success or -1 for error. If 'str' is not supplied the Fill with String dialog will appear when the macro is run.
Find()	int Find(string str) Searches for string 'str'. Returns TRUE if found, FALSE if not found, -1 for error. If 'str' is not supplied the Find dialog will appear when the macro is run. Find() will use the current settings on the Find dialog, but it will force the Perl Regular Expressions option OFF, and the Match Case option ON. See also Findi(), FindRE() and FindREi().
FindADiskFile	Issues the Find a Disk File command
FindAndCount()	int FindAndCount(string str) Searches for occurrences of 'str' and returns the number found. If 'str' is not supplied the Find and Count dialog will appear when the macro is run.
FindDifferingLines	int FindDifferingLines() Issues the Find Differing Lines command and returns 1 if a mismatch is found, or 0 if no additional mismatches exist.
FindDistinctLines	Issues the Find Distinct Lines command
FindDuplicateLines	Issues the Find Duplicate Lines command
FindFast	Issues the Find Fast command
Findi()	int Findi(string str) Searches for string 'str'. Returns TRUE if found, FALSE if not found, -1 for error. If 'str' is not supplied the Find dialog will appear when the macro is run. Findi() will use the current settings on the Find dialog, but it will force the Perl Regular Expressions option OFF, and the Match Case option OFF. See also Find(), FindRE() and FindREi().
FindMate	int FindMate search for a mate to the parenthetical sequence at the text cursor. Returns TRUE if found, FALSE if not found.
FindNext	int FindNext Returns 1 if the string is found, 0 if not found, -1 for

	error.
FindPrevious	int FindPrevious Returns 1 if the string is found, 0 if not found, -1 for error.
FindRE()	int FindRE(string str) Searches for string 'str'. Returns TRUE if found, FALSE if not found, -1 for error. If 'str' is not supplied the Find dialog will appear when the macro is run. FindRE() will use the current settings on the Find dialog, but it will force the Perl Regular Expressions option ON, and the Match Case option ON. See also Find(), Findi() and FindREi().
FindREi()	int FindREi(string str) Searches for string 'str'. Returns TRUE if found, FALSE if not found, -1 for error. If 'str' is not supplied the Find dialog will appear when the macro is run. FindREi() will use the current settings on the Find dialog, but it will force the Perl Regular Expressions option ON, and the Match Case option OFF. See also Find(), Findi() and FindRE().
FindTextInDiskFiles	Issues the Find Text in Disk Files command
FindUniqueLines	Issues the Find Unique Lines command
FlipCase	Issues the Flip Case command
floor()	float floor(float x) Returns (as a float) the largest integer not greater than 'x'. Example: floor(1.5) returns 1.0.
FormatXML	Issues the Format XML command
Formfeed	Issues the Formfeed command
FTPOpen()	int FTPOpen(string fn) Opens the FTP file 'fn' for editing. If 'fn' is already open for editing, its window will become the current window. Returns 1 for success or -1 for error. Remember: \\ must be used to denote \ within 'fn'.
GetChar()	int GetChar(string prompt, char c) Displays a message box with 'prompt' and fills 'c' with the character entered by the user. Returns 1 for success or -1 for error. See also PressChar().
GetClipboardText()	int GetClipboardText(string str, int n) Fills 'str' with the text of Clipboard 'n'. Returns the

	length of the text installed or -1 for error. The Windows Clipboard is number 0; private clipboards are numbered 1 to 8.
GetCurrentDirectory()	int GetCurrentDirectory(string str) Retrieves the current directory for the active process and places it in 'str'. Returns 1 for success or -1 for error. See also SetCurrentDirectory().
GetDataDirectory()	int GetDataDirectory(string str) Fills 'str' with the full path of the data directory. Returns 1 for success or -1 for error. See also GetProgramDirectory().
GetDate()	int GetDate(int y, int m, int d) Gets the current date and fills 'y', 'm' and 'd' with the year, month and date, respectively. Returns 1 for success or -1 for error.
GetDayName()	int GetDayName(string str, int n) Fills 'str' with the 3-character name of weekday number 'n' (1-7). The string returned is sensitive to the local language. Returns 1 for success or -1 for error.
GetEditMode()	int GetEditMode() Returns the edit mode of the current file. Returns 0 if the edit mode is Insert, or 1 if the edit mode is Typeover. Returns -1 if a file is not open.
GetEnv()	int GetEnv(string str1, string str2) Fills 'str1' with the content of the environment variable named in 'str2'. Returns 1 for success or -1 for error.
GetFloat()	int GetFloat(string prompt, float x) Displays a message box with 'prompt' and fills 'x' with the value entered by the user. Returns 1 for success or -1 for error.
GetGMTDateTime()	int GetGMTDateTime(int y, int m, int d, int hh, int mm, int ss) Gets the current date and time at GMT (Greenwich Mean Time) and fills 'y', 'm', 'd', 'hh', 'mm' and 'ss' with year/month/day/hour/minute/second, respectively. Hours will be in 24-hour format. Returns 1 for success or -1 for error.
GetInt()	int GetInt(string prompt, int n) Displays a message box with 'prompt' and fills 'n' with the value entered by the user. Returns 1 for success or -1 for error.

GetLineText()	int GetLineText(int n, string str) Fills 'str' with the text of line 'n'. Returns the length of line 'n' or -1 for error.
GetMonName()	int GetMonName(string str, int n) Fills 'str' with the 3-character name of month number 'n' (1-12). The string returned is sensitive to the local language. Returns 1 for success or -1 for error.
GetMonthName()	int GetMonthName(string str, int n) Fills 'str' with the full name of month number 'n' (1-12). The string returned is sensitive to the local language. Returns 1 for success or -1 for error.
GetProgramDirectory()	int GetProgramDirectory(string str) Fills 'str' with the full path of the program directory. Returns 1 for success or -1 for error. See also GetDataDirectory().
GetReadOnly	int GetReadOnly Returns the read-only state of the current file. Returns 1 if the file is read-only, 0 if the file is not read-only, or -1 for error. See also: ToggleReadOnly()
GetSelection()	int GetSelection(string str) Fills 'str' with the currently selected text. Returns the length of the selection or -1 for error. See also: ActiveClipboard
GetSelectionBounds()	int GetSelectionBounds(int l1, int c1, int l2, int c2) Fills l1, c1, l2, c2 with the bounds of the currently selected text. Returns 1 for success or -1 for error.
GetSelectionMode()	int GetSelectionMode() Returns 0 if the current selection mode is Stream, 1 if the current selection mode is Columnar
GetSelectionSize	int GetSelectionSize Returns the number of character currently selected, 0 if a selection is not present, or -1 for error.
GetShortName()	int GetShortName(string shortname, string fullname) Fills 'shortname' with the 8.3/DOS format short filename that corresponds to 'longname'. Returns 1 for success, or -1 for error.
GetString()	int GetString(string prompt, string result [, string default]) Displays a message box with 'prompt' and fills 'result' with the string entered by the user. If the

	optional third parameter 'default' is present, it is suggested as the default entry string. Returns the length of 'result' or -1 for error.
GetTextWidth	int GetTextWidth Returns the current Text Width value.
GetTime12()	int GetTime12(int h, int m, int s, int pm) Gets the current time and fills 'h', 'm' and 's' with hours, minutes and seconds, respectively. Hours will be in 12-hour format. If the time is PM, 'pm' is set to 1, else it is set to 0. Returns 1 for success or -1 for error.
GetTime24()	int GetTime24(int h, int m, int s) Gets the current time and fills 'h', 'm' and 's' with hours, minutes and seconds, respectively. Hours will be in 24-hour format. Returns 1 for success or -1 for error.
GetWeekday()	int GetWeekday(int y, int m, int d) Returns the number of the weekday associated with the date 'y', 'm', 'd'. Returns 1-7 for success or -1 for error.
GetWeekdayName()	int GetWeekdayName(string str, int n) Fills 'str' with the full name of weekday number 'n' (1-7). The string returned is sensitive to the local language. Returns 1 for success or -1 for error.
GetWindowNumber()	int GetWindowNumber(string fn) Returns the window number that holds the file 'fn'. Returns -1 for error, 0 if the named file is not open, or a postive value if the file's window is located. See also Filename(), SwitchToWindow().
GetWord()	int GetWord(string str) Fills 'str' with the word at the text cursor. Returns the length of the word found or -1 for error. See also SelectWord().
GetWordDelimiters()	int GetWordDelimiters(string str) Fills 'str' with a string that contains the characters considered to be word delimiters for the current file. Returns 1 for success or -1 for error.
GetYesNo()	int GetYesNo(string title, string query) Gets a Yes or No reply from the user. Displays a message box with title 'title' and message 'query'. Returns 1 if the user clicks Yes, 0 if the user clicks No.
GoToByteOffset()	int GoToByteOffset(int n OR string str) Go to offset 'n' in the current file. Returns 1 for

	success or -1 for error. A string parameter is also accepted. For example: "+25" will cause the cursor to be moved ahead 25 bytes. If a parameter is not provided the Go to Byte Offset dialog will appear when the macro is run.
GoToColumn()	int GoToColumn(int n OR string str) Go to column 'n' in the current file. Returns 1 for success or -1 for error. A string parameter is also accepted. For example: "-12" will cause the cursor to be moved 12 columns to the left. If a parameter is not provided the Go to Column dialog will appear when the macro is run.
GoToLine()	int GoToLine(int n or string str) Go to line 'n' in the current file. Returns 1 for success or -1 for error. A string parameter is also accepted. For example: "+50" will cause the cursor to be moved ahead 50 lines. If a parameter is not provided the Go to Line dialog will appear when the macro is run.
GoToParagraph()	int GoToParagraph(int n) Go to paragraph 'n' in the current file. Returns 1 for success or -1 for error. If a parameter is not provided the Go to Paragraph dialog will appear when the macro is run.
HardenLineEnders	Issues the Harden Line Enders command.
HTMLImageTag()	int HTMLImageTag(string fn) Inserts an HTML 'IMG' tag for the image file 'fn'. BMP, GIF and JPG images are supported. Returns 1 for success or -1 for error.
Increment()	int Increment(int n) Adds 'n' to the value at the text cursor and places the result in the text file. Returns the result of the operation or -1 for error. If 'n' is not supplied the Increment dialog will appear when the macro is run.
IndentOneSpace	Issues the Indent One Space command
IndentOneTabstop	Issues the Indent One Tabstop command
IndentWithString()	int IndentWithString(string str) Indents the selected lines with 'str'. Returns 1 for success or -1 for error. If 'str' is not supplied the Indent with String dialog will appear when the macro is run.
InsertCharacter()	int InsertCharacter(char ch) Inserts character 'ch' into the edited text. Returns the ASCII value of 'ch' or -1 for error. (This command is identical to PutChar.)

InsertFile()	int InsertFile(string str) Insert file 'str' into the current file. Returns 1 for success or -1 for error. If 'str' is not provided the Insert File dialog will appear when the macro is run.
InsertFilename	Issues the Insert Filename command
InsertLineAbove	Issues the Insert Line Above command
InsertLineBelow	Issues the Insert Line Below command
InsertLongDate	Issues the Insert Long Date command
InsertLongTime	Issues the Insert Long Time command
InsertMode	InsertMode Switches the edit mode to Insert in the current file. See also ToggleEditMode().
InsertShortDate	Issues the Insert Short Date command
InsertShortTime	Issues the Insert Short Time command
InvertLines	Issues the Invert Lines command
isalnum()	int isalnum(char c) Returns non-zero if character 'c' is alphanumeric.
isalpha()	int isalpha(char c) Returns non-zero if character 'c' is alphabetic.
isascii()	int isascii(char c) Returns non-zero if character 'c' is in the range 0-127.
IsBookmarked()	int IsBookmarked(int n) Returns 1 if line 'n' is bookmarked, 0 if not, or -1 for error. If 'n' is not supplied, the current line is assumed.
iscntrl()	int iscntrl(char c) Returns non-zero if character 'c' is a control character (0-31 or 127).
isdigit()	int isdigit(char c) Returns non-zero if character 'c' is a digit.
islower()	int islower(char c) Returns non-zero if character 'c' is lowercase.
ispunct()	int ispunct(char c) Returns non-zero if character 'c' is punctuation.
isspace()	int isspace(char c) Returns non-zero if character 'c' is whitespace (space, tab, newline, etc.).
isupper()	int isupper(char c) Returns non-zero if character 'c' is uppercase.

isxdigit()	int isxdigit(char c) Returns non-zero if character 'c' is a hex digit (A-F, a-f, 0-9).
JustificationStyle()	int JustificationStyle(int n) Sets the current text justification style according to 'n': 1=Left, 2=Center, 3=Right, 4=Smooth. If 'n' is not supplied the Justification Style dialog will appear when the macro runs. Returns 1 for success or -1 for error.
LastCharacter()	int LastCharacter(string str) Returns the last character in 'str' or 0 if 'str' is an empty string.
Left	int Left(int n) Issues the Left command 'n' times. Returns the number of commands performed or -1 for error. The argument 'n' is optional; if it is not provided a single command is performed.
LeftWindowEdge	Issues the Left Window Edge command
LineContains()	int LineContains(int n, string str) Returns the number of occurrences of 'str' that appear in line 'n'. The search performed is case sensitive. Regular expressions are not recognized.
LineContainsi()	int LineContainsi(int n, string str) Returns the number of occurrences of 'str' that appear in line 'n'. The search performed is case insensitive. Regular expressions are not recognized.
LineContainsRE()	int LineContainsRE(int n, string str) Returns the number of matches to 'str' that appear in line 'n'. The search performed is case sensitive. Regular expressions ARE recognized.
LineContainsREi()	int LineContainsREi(int n, string str) Returns the number of matches to 'str' that appear in line 'n'. The search performed is case insensitive. Regular expressions ARE recognized.
LineCount	int LineCount Returns the number of lines in the current file.
LineDrawing()	int LineDrawing(int style) Initiates or terminates Line Drawing mode. If 'style' is 1 to 11, a corresponding line style from the Line Drawing dialog is activated. The Up, Down, Left and Right commands can then be used to draw lines and boxes. When 'style' is 0, Line Drawing mode is terminated. If 'style' is not supplied the Line Drawing dialog will appear when the macro runs. Returns 1 for success or -1 for error.

LineIsEmpty()	int LineIsEmpty(int n) Returns TRUE if line 'n' is empty. Note: a line containing only whitespace is considered empty.
LineLength()	int LineLength(int n) Returns the number of characters in line 'n'.
LineNumber	int LineNumber Returns the current line number in the current file or -1 for error.
LineSpacing	int LineSpacing(int mode) Formats the range of selected lines, or the whole file, according to 'mode'. 'mode' can be 1, 2 or 3, which produces single, double or triple spacing, respectively. Returns 1 for success or -1 for error.
log()	float log(float x) Returns the natural log of 'x'. 'x' must be a positive value greater than 0.
log10()	float log(float x) Returns the base 10 log of 'x'. 'x' must be a positive value greater than 0.
MakeLineBottom	Issues the Make Line Bottom command
MakeLineCenter	Issues the Make Line Center command
MakeLineTop	Issues the Make Line Top command
max()	int max(int n1, int n2) Returns the greater of 'n1' and 'n2'.
Maximize	Maximize the current editing window.
MaximizeAll	Issues the Maximize All command
Message()	Message(string str, ...) Displays a pop-up message box with title 'str' and a message that is built from all arguments that follow. Example: Message("Results", n, " removed;", m, " remain.");
min()	int min(int n1, int n2) Returns the lesser of 'n1' and 'n2'.
Minimize	Minimize the current editing window.
MinimizeAll	Issues the Minimize All command
Modified	int Modified Returns 1 if the current file has been modified, else 0. Returns -1 for error. See also SetModified().
MoveLineDown	Issues the Move Line Down command

MoveLineUp	Issues the Move Line Up command
Multiply()	int Multiply(int n) Multiplies the value at the text cursor by 'n' and places the result in the text file. Returns the result of the operation or -1 for error. If 'n' is not supplied the Multiply dialog will appear when the macro is run.
New	Issues the New command
NextBookmark	Issues the Next Bookmark command
NextFunction	Issues the Next Function command
NextParagraph	Issues the Next Paragraph command
OEMChart	Issues the OEM Chart command
OEMtoANSI	Issues the OEM to ANSI command
Open()	int Open(string fn) Opens the file 'fn' for editing. If 'fn' is already open for editing, its window will become the current window. Returns 1 for success or -1 for error. Remember: \\ must be used to denote \ within 'fn'.
OpenEmail()	int OpenEmail(string str) Initiates an email message to the address in 'str' using the default email client. Returns 1 for success or -1 for error. See also OpenEmailAtCursor.
OpenEmailAtCursor	Issues the Open Email at Cursor command
OpenFileInBrowser	Issues the Open File in Browser command
OpenFilenameAtCursor	Issues the Open Filename at Cursor command
OpenHeaderFile	Issues the Open Header File command
OpenHex()	int OpenHex(string fn) Opens the file 'fn' for hex mode viewing and editing. Returns 1 for success or -1 for error. Remember: \\ must be used to denote \ within 'fn'.
OpenProgramAtCursor	Issues the Open Program at Cursor command
OpenRecentFile()	OpenRecentFile(int n) Opens recent file number 'n'. When a sufficient file history exists, 'n' can range from 1 to 24.
OpenRecentProject()	OpenRecentProject(int n) Opens recent project number 'n'. When a sufficient project history exists, 'n' can range from 1 to 16.
OpenSystemFiles	Issues the Open System Files command
OpenURL()	int OpenURL(string str)

	Opens the URL described in 'str' in the default internet browser. Returns 1 for succes or -1 for error. See also OpenURLAtCursor.
OpenURLAtCursor	Issues the Open URL at Cursor command
PageDown	Issues the Page Down command
PageLeft	Issues the Page Left command
PageRight	Issues the Page Right command
PageSetup	Issues the Page Setup command
PageUp	Issues the Page Up command
Paste	Issues the Paste command
PasteAs	Issues the Paste As command
PasteClipboard()	PasteClipboard(int n) Pastes the content of Clipboard 'n' into the current file. The Windows Clipboard is number 0; private clipboards are numbered 1 to 8.
Pause	Pause Pauses macro execution by displaying a message box and waiting for it to be closed.
pi	float pi Returns the value of pi, which is approximately 3.1415926536.
PlaySound()	int PlaySound(string filepath) Plays the .WAV file described in 'filepath'. Returns 1 for success or -1 for error.
pow()	float pow(float x, float y) Returns the value of 'x' raised to the power 'y'.
PressChar()	int PressChar(string prompt, char c) Displays the message 'prompt' on the status bar and fills 'c' with the next character pressed by the user. A popup dialog does NOT appear. A file must be open in order for PressChar to operate. Returns 1 for success or -1 for error. Note: PressChar will not wait for a character when run in Debug mode. See also GetChar().
PreviousBookmark	Issues the Previous Bookmark command
PreviousFunction	Issues the Previous Function command
PreviousParagraph	Issues the Previous Paragraph command

Print	Issues the Print command
PrintAll	Issues the Print All command
PrintAllColor	Issues the Print All Color command
PrintAllMonochrome	Issues the Print All Monochrome command
PrintColor	Issues the Print Color command
printf()	int printf(string format, ...) Processes 'format' and inserts a string into the edited text, in accordance with the formatting commands used in 'C'. Returns the number of characters inserted. See the online help for more information.
PrintMonochrome	Issues the Print Monochrome command
PrintPreview	Issues the Print Preview command
PrintPreviewColor	Issues the Print Preview Color command
PrintPreviewMonochrome	Issues the Print Preview Monochrome command
PrintSetup	Issues the Print Setup command
ProjectAddAll	Issues the Project Add All command
ProjectAddOne	Issues the Project Add One command
ProjectAutoUpdate()	int ProjectAutoUpdate(int mode) Toggles the Auto-Update feature on or off for the active project according to 'mode'. Returns 1 for success or -1 for error.
ProjectClose	Issues the Project Close command
ProjectDelete()	int ProjectDelete(string name) Deletes the project file described by 'name'. A confirmation prompt will be presented. Returns 1 for success or -1 for error.
ProjectEditActive	Issues the Project Edit Active command
ProjectEditOther()	int ProjectEditOther(string name) Opens the project file described by 'name' for editing. If 'name' does not exist an empty file will be opened. Returns 1 for success or -1 for error.
ProjectName()	int ProjectName(string fn) Fills 'fn' with the full path of the active project file. Returns the length of the filepath or -1 for error.
ProjectNew	Issues the Project New command
ProjectOpen()	int ProjectOpen(string name) Open the project file described by 'name'. Returns 1 for success or -1 for error.
ProjectRemove	Issues the Project Remove command

ProjectUpdateAll	Issues the Project Update All command
ProjectUpdateOne	Issues the Project Update One command
PutChar()	int PutChar(char ch) Inserts character 'ch' into the edited text. Returns 1 for success or -1 for error.
PutClipboardText()	int PutClipboardText(string str, int n) Fills Clipboard 'n' with string 'str'. Returns the length of the text installed or -1 for error. The Windows Clipboard is number 0; private clipboards are numbered 1 to 8. See also AppendToClipboard().
PutFloat()	int PutFloat(float x) Inserts the value of 'x' into the edited text. Two decimal places will be used. Returns 1 for success or -1 for error. Use printf() if special formatting is required.
PutInt()	int PutInt(int n) Inserts the value of 'n' into the edited text. Returns 1 for success or -1 for error.
PutLineText()	int PutLineText(int n, string str) Replaces the text of line 'n' with 'str'. Returns the length of 'str' or -1 for error.
PutMany()	int PutMany(...) Inserts the supplied argument(s) into the edited text. Example: PutMany(5, " is a number", '\n');
PutSelection()	int PutSelection(string str) Inserts string 'str' into the edited text. Returns the length of 'str' or -1 for error. PutSelection() is the complement to GetSelection(), and it should be used instead of PutString to ensure proper insertion of column-selected text.
PutString()	int PutString(string str) Inserts string 'str' into the edited text. Returns the length of 'str' or -1 for error.
PutWordDelimiters()	int PutWordDelimiters(string str) Sets the word delimiters for the current file to the characters contained in 'str'. Returns 1 for success or -1 for error.
QuoteAndReformat	Issues the Quote and Reformat command
Random()	int Random(int n) Returns a random number between 0 and n-1 or -1 for error.

ReadValue()	int ReadValue(string name, char/int/string/float val) Reads a value from the macro variable storage area named 'name' and places it into variable 'val'. The type of 'val' must agree with the type used when the value was written using WriteValue(). Returns 1 for success or -1 for error. See also WriteValue(), EraseValue(), ValueExists().
Redo	Issues the Redo command
RedoAll	Issues the Redo All command
Reference	Issues the Reference command
Reformat	Issues the Reformat command
ReloadFile	Issues the Reload File command
RenameFile()	int RenameFile(string oldname, string newname) Renames the file or directory named 'oldname' to 'newname'. Files can be renamed across drives; directories must be on the same drive. The target 'newname' must not exist. Returns 1 for success or -1 for error.
Replace()	int Replace(string str1, string str2) Searches for 'str1' and replaces it with 'str2'. Returns the number of replacements made or -1 for error. The user will be prompted to confirm replacements. If 'str1' and 'str2' are not supplied the Replace dialog will appear when the macro is run. Replace() will use the current settings on the Replace dialog, but it will force the Perl Regular Expressions option OFF, and the Match Case option ON. See also Replacei().
ReplaceAgain	Issues the Replace Again command
ReplaceAll()	int ReplaceAll(string str1, string str2) Searches for 'str1' and replaces it with 'str2'. Returns the number of replacements made or -1 for error. The user will NOT be prompted to confirm replacements. ReplaceAll() will use the current settings on the Replace dialog, but it will force the Perl Regular Expressions option OFF, and the Match Case option ON. See also ReplaceAlli().
ReplaceAlli()	int ReplaceAlli(string str1, string str2) Searches for 'str1' and replaces it with 'str2'. Returns the number of replacements made or -1 for error. The user will NOT be prompted to confirm

	replacements. ReplaceAlli() will use the current settings on the Replace dialog, but it will force the Perl Regular Expressions option OFF, and the the Match Case option OFF. See also ReplaceAll().
ReplaceAllRE()	int ReplaceAllRE(string str1, string str2) Searches for 'str1' and replaces it with 'str2'. Returns the number of replacements made or -1 for error. The user will be prompted to confirm replacements. If 'str1' and 'str2' are not supplied the Replace dialog will appear when the macro is run. ReplaceAllRE() will use the current settings on the Replace dialog, but it will force the Perl Regular Expressions option to ON, and the Match Case option ON. See also ReplaceAll(), ReplaceAlli and ReplaceAllREi().
ReplaceAllREi()	int ReplaceAllREi(string str1, string str2) Searches for 'str1' and replaces it with 'str2'. Returns the number of replacements made or -1 for error. The user will be prompted to confirm replacements. If 'str1' and 'str2' are not supplied the Replace dialog will appear when the macro is run. ReplaceAllREi() will use the current settings on the Replace dialog, but it will force the Perl Regular Expressions option to ON, and the Match Case option OFF. See also ReplaceAll(), ReplaceAlli and ReplaceAllRE().
Replacei()	int Replacei(string str1, string str2) Searches for 'str1' and replaces it with 'str2'. Returns the number of replacements made or -1 for error. The user will be prompted to confirm replacements. If 'str1' and 'str2' are not supplied the Replace dialog will appear when the macro is run. Replacei() will use the current settings on the Replace dialog, but it will force the Perl Regular Expressions option OFF, and the Match Case option OFF. See also Replace().
ReplaceLineEnders()	int ReplaceLineEnders(string str1, string str2) Searches for 'str1' and replaces it with 'str2'. 'str1' and 'str2' may use the sequence \n (if within a quoted string) to represent a line ender. Returns the number of replacements made or -1 for error. The

	user will be NOT prompted to confirm replacements. If 'str1' and 'str2' are not supplied the Replace Line Enders dialog will appear when the macro is run. ReplaceLineEnders() will use the current settings on the Replace Line Enders dialog.
ReplaceRE()	int ReplaceRE(string str1, string str2) Searches for 'str1' and replaces it with 'str2'. Returns the number of replacements made or -1 for error. The user will be prompted to confirm replacements. If 'str1' and 'str2' are not supplied the Replace dialog will appear when the macro is run. ReplaceRE() will use the current settings on the Replace dialog, but it will force the Perl Regular Expressions option to ON, and the Match Case option ON. See also Replace(), Replacei() and ReplaceREi().
ReplaceREi()	int ReplaceREi(string str1, string str2) Searches for 'str1' and replaces it with 'str2'. Returns the number of replacements made or -1 for error. The user will be prompted to confirm replacements. If 'str1' and 'str2' are not supplied the Replace dialog will appear when the macro is run. ReplaceREi() will use the current settings on the Replace dialog, but it will force the Perl Regular Expressions option to ON, and the Match Case option OFF. See also Replace(), Replacei() and ReplaceRE().
Restore	Restore the current window from a minimized or maximized state.
RestoreAll	Issues the Restore All command
RestoreSettings	RestoreSettings Restores a variety of editor settings which were earlier noted using SaveSettings(). These functions can be used to ensure that a macro does not alter the editor's settings. The following settings are restored: Wordrwrap, Text Width, Justification Style, Edit Mode, Tab Display Size, Selection Mode, Active Clipboard and Word Delimiters.
Right	int Right(int n) Issues the Right command 'n' times. Returns the number of commands performed or -1 for error. The argument 'n' is optional; if it is not provided a single command is performed.
RightWindowEdge	Issues the Right Window Edge command
ROT5	Applies a ROT5 (rotation 5) conversion to the

	selected text
ROT13	Applies a ROT13 (rotation 13) conversion to the selected text
ROT18	Applies a ROT18 (rotation 18) conversion to the selected text
ROT47	Applies a ROT47 (rotation 47) conversion to the selected text
Round()	float Round(float x, int n) Returns the value of 'x' rounded to 'n' decimal places.
RunProgram()	int RunProgram(string fn, string params, string workdir, int wait) Runs the named program, document or folder in 'fn' by passing it to the ShellExecuteEx Windows API call. Command line parameters can be passed in 'params'. The program's working directory can be passed in 'workdir'. If 'wait' is 1, macro execution will be suspended until the program has been closed. Returns the completion code of ShellExecuteEx: non-zero for success, zero for error. See also OpenProgramAtCursor().
Save	Issues the Save command
SaveACopyAs()	int SaveACopyAs(string fn) Saves a copy of the current file to the file 'fn'. The name of the current file is not changed. Returns 1 for success or -1 for error. Remember: \\ must be used to denote \ within 'fn'.
SaveAll	Issues the Save All command
SaveAs()	int SaveAs(string fn) Saves the current file to the file 'fn'. The name of the current file is changed to 'fn'. Returns 1 for success or -1 for error. Remember: \\ must be used to denote \ within 'fn'.
SaveSelectionAs()	int SaveSelectionAs(string fn) Saves the current selection to the file 'fn'. Returns 1 for success or -1 for error. Remember: \\ must be used to denote \ within 'fn'.
SaveSettings	SaveSettings Records a variety of editor settings for later restoration using RestoreSettings(). These functions can be used to ensure that a macro does not alter the editor's settings. The following settings are saved: Wordrwrap, Text Width, Justification Style, Edit Mode, Tab Display Size, Selection Mode, Active

	Clipboard and Word Delimiters.
ScrollDown	Issues the Scroll Down command
ScrollLeft	Issues the Scroll Left command
ScrollRight	Issues the Scroll Right command
ScrollUp	Issues the Scroll Up command
SelectAllText	Issues the Select All Text command
SelectColumnar	Issues the Select Columnar command
SelectDown	Issues the Select Down command
SelectLeft	Issues the Select Left command
SelectPageDown	Issues the Select Page Down command
SelectPageLeft	Issues the Select Page Left command
SelectPageRight	Issues the Select Page Right command
SelectPageUp	Issues the Select Page Up command
SelectRight	Issues the Select Right command
SelectStream	Issues the Select Stream command
SelectToBottomOfPage	Issues the Select to Bottom of Page command
SelectToEndOfFile	Issues the Select to End of File command
SelectToEndOfLine	Issues the Select to End of Line command
SelectToStartOfFile	Issues the Select to Start of File command
SelectToStartOfLine	Issues the Select to Start of Line command
SelectToTopOfPage	Issues the Select to Top of Page command
SelectUp	Issues the Select Up command
SelectWithoutShift	Issues the Select without Shift command
SelectWord	int SelectWord(string str) Selects the word at the text cursor and places it in 'str'. Returns the length of the word selected or 0 if no word could be found to select. See also GetWord().
SelectWordLeft	Issues the Select Word Left command
SelectWordRight	Issues the Select Word Right command
SetClipboard()	SetClipboard(int n) Sets the active clipboard to Clipboard 'n'. The Windows Clipboard is number 0; private clipboards are numbered 1 to 8.
SetClipboardNext	Issues the Set Clipboard Next command

SetClipboardPrevious	Issues the Set Clipboard Previous command
SetCurrentDirectory	int SetCurrentDirectory(string str) Sets the current directory for the active process to 'str'. Returns 1 for success or -1 for error. See also GetCurrentDirectory().
SetModified	SetModified Sets the modifed state of the current to true. This might be used to force a Save operation even when a file has not been modified. Returns 1 for success or -1 for error. See also Modified().
ShadedTabZones()	int ShadedTabZones(int mode) Enables or disables the Shaded Tab Zones display mode according to 'mode'.
sin()	float sin(float x) Returns the sine of 'x'. The angle 'x' must be in radians.
sinh()	float sinh(float x) Returns the hyperbolic sine of 'x'. The angle 'x' must be in radians.
SoftenLineEnders	Issues the Soften Line Enders command.
SortFileTabsByExt()	int SortFileTabsByExt(int mode) Enables or disables the sorting of File Tabs by extension according to 'mode'. If 'mode' is 1 the feature is enabled. If 'mode' is 0 the feature is disabled.
SortFileTabsByName()	int SortFileTabsByName(int mode) Enables or disables the sorting of File Tabs by name according to 'mode'. If 'mode' is 1 the feature is enabled. If 'mode' is 0 the feature is disabled.
SortFileTabsByUse()	int SortFileTabsByUse(int mode) Enables or disables the sorting of File Tabs by name according to 'mode'. If 'mode' is 1 the feature is enabled. If 'mode' is 0 the feature is disabled.
SortLines	Issues the Sort Lines command
Space	Issues the Space command. If a range of lines is selected, the range will be indented.
SpacesToTabs	Issues the Spaces to Tabs command
SpellChecker	Issues the Spell Checker command
SplitHorizontal	Issues the Split Horizontal command
SplitVertical	Issues the Split Vertical command

sprintf()	int sprintf(string str1, string format, ...) Processes 'format' and builds an output string in 'str1', in accordance with the formatting commands used in 'C'. Returns the length of 'str1' or -1 for error. See the online help for more information.
sqrt()	float sqrt(float x) Returns the positive square root of 'x'.
StartOfFile	Issues the Start of File command
StartOfLine	Issues the Start of Line command
StatusMessage()	StatusMessage(...) Displays a message in the status bar that is built from all arguments that follow. Example: StatusMessage(n, " were removed;", m, " remain.");
strcat()	int strcat(string str1, string str2) Concatenates 'str2' to 'str1'. Returns the length of 'str1' or -1 for error.
strchr()	int strchr(string str, char c) Returns the offset at which the character 'c' appears in 'str' or -1 if 'c' does not appear. See also strrchr().
strcmp()	int strcmp(string str1, string str2) Compares 'str1' to 'str2' with case sensitivity. Returns 0 if the strings are equal. Returns < 0 if 'str1' is less than 'str2'. Returns > 0 if 'str1' is greater than 'str2'.
strcmpi()	int strcmpi(string str1, string str2) Compares 'str1' to 'str2' without case sensitivity. Returns 0 if the strings are equal. Returns < 0 if 'str1' is less than 'str2'. Returns > 0 if 'str1' is greater than 'str2'.
strcpy()	int strcpy(string str1, string str2) Copies 'str2' to 'str1'. Returns the length of 'str1' or -1 for error.
StripHTMLTags	Issues the Strip HTML/XML Tags command
StripLeadingSpaces	Issues the Strip Leading Spaces command
StripTrailingSpaces	Issues the Strip Trailing Spaces command
strlen()	int strlen(string str) Returns the length of 'str'.
strlwr()	int strlwr(string str) Converts the string 'str' to lowercase. Returns the length of 'str'.
strncat()	int strncat(string str1, string str2, int n)

	Concatenates up to 'n' characters from 'str2' to 'str1'. Returns the length of 'str1' or -1 for error. A NULL byte is added after the characters added.
strncmp()	int strncmp(string str1, string str2, int n) Compares up to 'n' characters in 'str1' to 'str2' with case sensitivity. Returns 0 if the strings are equal. Returns < 0 if 'str1' is less than 'str2'. Returns > 0 if 'str1' is greater than 'str2'.
strncmpi()	int strncmpi(string str1, string str2, int n) Compares up to 'n' characters in 'str1' to 'str2' without case sensitivity. Returns 0 if the strings are equal. Returns < 0 if 'str1' is less than 'str2'. Returns > 0 if 'str1' is greater than 'str2'.
strncpy()	int strncpy(string str1, string str2, int n) Copies up to 'n' characters from 'str2' to 'str1'. Returns the length of 'str1' or -1 for error. A NULL byte is added after the characters copied.
strrchr()	int strrchr(string str, char c) Returns the offset at which the character 'c' last appears in 'str' or -1 if 'c' does not appear. See also strchr().
strrev()	int strrev(string str) Reverses the string 'str' in place. Example: The string 'Boxer' would be converted to 'rexoB'.
strstr()	int strstr(string str1, string str2) Searches string 'str1' for the substring 'str2' with case sensitivity. Returns the offset at which 'str2' is found or -1 if not found.
strstri()	int strstri(string str1, string str2) Searches string 'str1' for the substring 'str2' without case sensitivity. Returns the offset at which 'str2' is found or -1 if not found.
strupr()	int strupr(string str) Converts the string 'str' to uppercase. Returns the length of 'str'.
SubString()	int SubString(string str1, string str2, int index, int len) Fills 'str1' with up to 'len' characters from 'str2', starting at offset 'index'. The first character in a string is referred to by offset 0. Returns the new length of 'str1' or -1 for error. If 'index' is negative, and 'len' is positive, 'str1' will be filled with 'len' characters starting 'index' characters in from the end of 'str2'.

	If 'len' is negative, the value of 'index' is ignored, and 'str1' is filled with the rightmost 'len' characters from 'str2'.
SwapLines	Issues the Swap Lines command
SwapWords	Issues the Swap Words command
SwitchToWindow()	SwitchToWindow(int n) Makes window 'n' active. Windows are numbered from 1 to the number of open files. See also FileCount. See also GetWindowNumber().
SyntaxHighlightAs	Issues the Syntax Highlight As command
Tab	Issues the Tab command. If a range of lines is selected, the range will be indented.
TabDisplaySize()	int TabDisplaySize(string str) Sets the Tab Display Size for the current file according to 'str'. If a single value appears in 'str', tabs are set to fixed width with the value in 'str'. If a series of comma-separated values are found in 'str', variable width tab stops will be used. Returns TRUE for success, -1 for error. If 'str' is not supplied the Tab Display Size dialog will appear when the macro is run.
TabsToSpaces	Issues the Tabs to Spaces command
tan()	float tan(float x) Returns the tangent of 'x'. The angle 'x' must be in radians.
tanh()	float tanh(float x) Returns the hyperbolic tangent of 'x'. The angle 'x' must be in radians.
Templates	Issues the Templates command
TextIsSelected	int TextIsSelected Returns 1 if a stream selection is present, 2 if a columnar selection is present, or 0 if no selection is present. Returns -1 for error.
TextWidth()	int TextWidth(int n) Sets the Text Width to 'n'. Returns TRUE for success or -1 for error. If 'n' is not supplied the Text Width dialog will appear when the macro is run.
TileAcross	Issues the Tile Across command
TileDown	Issues the Tile Down command

ToggleBookmark()	int ToggleBookmark(int n, int state) Sets bookmark 'n' according to 'state'. Returns TRUE for success, -1 for error. If 'state' is ON, bookmark 'n' is placed on the current line. If 'state' is OFF bookmark 'n' is cleared, where ever it is.
ToggleEditMode()	ToggleEditMode(int state) Toggles the edit mode according to 'state'. If 'state' is 1, Insert mode is used. If 'state' is 0, Typeover mode is used. See also the functions InsertMode() and TypeoverMode().
ToggleReadOnly()	ToggleReadOnly(int state) Toggle read-only mode according to 'state'. If 'state' is 1, read-only mode is set. If 'state' is 0, read-only mode is released.
tolower()	int tolower(char c) Returns the lowercase mate to character 'c'.
ToolbarBottom	Issues the Toolbar Bottom command
ToolbarLeft	Issues the Toolbar Left command
ToolbarRight	Issues the Toolbar Right command
ToolbarTop	Issues the Toolbar Top command
TopLine	int TopLine Returns the line number of the first line in the editor window. Returns -1 for error.
TopOfPage	Issues the Top of Page command
TotalAndAverage	Issues the Total and Average command
TouchFile()	int TouchFile(string name) Touch (ie, update the timestamp of) the file named 'name'. Returns 1 for success or -1 for error.
toupper()	int toupper(char c) Returns the uppercase mate to character 'c'.
Trim()	int Trim(string str) Removes leading and trailing blanks from 'str'. Returns the new length of 'str'.
TrimLeft()	int TrimLeft(string str) Removes leading blanks from 'str'. Returns the new length of 'str'.
TrimRight()	int TrimRight(string str) Removes trailing blanks from 'str'. Returns the new length of 'str'.
Trunc()	float Trunc(float x, int n)

	Returns the value of 'x' truncated to 'n' decimal places.
TypeoverMode	TypeoverMode Switches the edit mode to Typeover in the current file. See also ToggleEditMode().
Uncomment	Issues the Uncomment command.
Undo	Issues the Undo command.
UndoAll	Issues the Undo All command.
UndoAllClosedTabs	Issues the Undo All Closed Tabs command.
UndoClosedTab()	UndoClosedTab(int n) Reopens recently closed file tab number 'n'. When a sufficient closed tab list exists, 'n' can range from 1 to 10.
UndoCloseTab	Issues the Undo Close Tab command.
Unformat	Issues the Unformat command.
UnformatXML	Issues the Unformat XML command.
UnhighlightMatches	Issues the Unhighlight Matches command.
Unindent	Issues the Unindent command.
Up	int Up(int n) Issues the Up command 'n' times. Returns the number of commands performed or -1 for error. The argument 'n' is optional; if it is not provided a single command is performed.
UserList()	UserList(int n) Opens User List window 'n'.
UserTool()	UserTool(int n) Runs User Tool number 'n'.
UserToolWait()	UserTool(int n) Runs User Tool number 'n' and waits for it to complete execution.
ValueAtCursor	int ValueAtCursor Returns the ASCII value of the character at the cursor, or -1 if a character is not available at the cursor.
ValueExists()	int ValueExists(string name) Looks for the variable 'name' in the macro variable storage area. Returns 1 if it exists, 0 if it does not exists, or -1 for error. See also ReadValue(), WriteValue(), EraseValue().

ViewBookmarks()	int ViewBookmarks(int mode) Enables or disables the viewing of Bookmarks according to 'mode'.
ViewFileTabs()	int ViewFileTabs(int mode) Enables or disables the viewing of File Tabs according to 'mode'.
ViewHexMode()	int ViewHexMode(int mode) Enables or disables read-only hex mode viewing according to 'mode'.
ViewHexRuler()	int ViewHexRuler(int mode) Enables or disables the viewing of the Hex Ruler according to 'mode'.
ViewTextRuler()	int ViewTextRuler(int mode) Enables or disables the viewing of the Text Ruler according to 'mode'.
ViewHScrollBar()	int ViewHScrollBar(int mode) Enables or disables the viewing of Horizontal Scroll Bars according to 'mode'.
ViewLineNumbers()	int ViewLineNumbers(int mode) Enables or disables the viewing of Line Numbers according to 'mode'.
ViewRightMarginRule()	int ViewRightMarginRule(int mode) Enables or disables the viewing of the Right Margin Rule according to 'mode'.
ViewStatusBar()	int ViewStatusBar(int mode) Enables or disables the viewing of the Status Bar according to 'mode'.
ViewSyntaxHighlighting()	int ViewSyntaxHighlighting(int mode) Enables or disables the Syntax Highlighting feature according to 'mode'.
ViewTextHighlighting()	int ViewTextHighlighting(int mode) Enables or disables the Text Highlighting feature according to 'mode'.
ViewToolbar()	int ViewToolbar(int mode) Enables or disables the viewing of the Toolbar according to 'mode'.
ViewVisibleSpaces()	int ViewVisibleSpaces(int mode) Enables or disables the viewing of Visible Spaces according to 'mode'.
ViewVScrollBar()	int ViewVScrollBar(int mode) Enables or disables the viewing of Vertical Scroll Bars according to 'mode'.
VisualWrap()	int VisualWrap(int mode) Enables or disables Visual Wrap according to 'mode'.

VisualWrapOptions	Issues the Visual Wrap Options command.
Wait()	int Wait(int n) Delays macro execution for 'n' milliseconds. 1000 milliseconds equals 1 second. Returns 1 for success or -1 for error.
WindowHeight	int WindowHeight Returns the number of lines that can be displayed in the current window.
WindowLastVisited	Issues the Window Last Visited command
WindowList	Issues the Window List command
WindowNext	Issues the Window Next command
WindowPrevious	Issues the Window Previous command
WindowSkip	int WindowSkip(int mode) If 'mode' is 1, sets the skip status for the current window to on. If mode is 0, skip status is turned off. Returns 1 for success or -1 for error.
WindowWidth	int WindowWidth Returns the number of columns that can be displayed in the current window.
WordCount()	int WordCount(int lines, int words, int chars) Fills the supplied variables with the count of lines, words and characters, respectively. Returns 1 for success or -1 for error.
WordLeft	int WordLeft(int n) Issues the Word Left command 'n' times. Returns the number of commands performed or -1 for error. The argument 'n' is optional; if it is not provided a single command is performed.
WordRight	int WordRight(int n) Issues the Word Right command 'n' times. Returns the number of commands performed or -1 for error. The argument 'n' is optional; if it is not provided a single command is performed.
WordWrap()	int WordWrap(int mode) Enables or disables Typing Wrap according to 'mode'. Deprecated: see TypingWrap().
WriteValue()	int WriteValue(string name, char/int/string/float val) Writes 'val' to the macro variable storage area named 'name'. 'name' will be visible to other macros, so be careful to choose a unique identifier. Returns 1 for success or -1 for error. See also ReadValue(), EraseValue(), ValueExists().

xtoi()	int xtoi(string str) Returns the integer value of the hexadecimal number described by string 'str'. Returns -1 for error.

4.7.4 Macro Examples

The following example macros show the syntax of Boxer's macro language, while also suggesting useful methods of attack for common programming tasks:

Move cursor to bottom of paragraph

```
// move the cursor to the bottom line of the current paragraph

macro BottomOfParagraph()
{
while (LineNumber < LineCount && !LineIsEmpty(LineNumber+1))
   Down;
}
```

Move cursor to top of previous paragraph

```
// move the cursor to the top line of the previous paragraph

macro TopOfPreviousParagraph()
{
Up;

while (LineNumber > 1 && !LineIsEmpty(LineNumber-1))
   Up;

StartOfLine;
}
```

Move cursor to top of current paragraph

```
// move the cursor to the top line of the current paragraph

macro TopOfCurrentParagraph()
{
while (LineNumber > 1 && !LineIsEmpty(LineNumber-1))
```

```
    Up;
}
```

Move cursor to top of next paragraph

```
// move the cursor to the first line of the next paragraph

macro TopOfNextParagraph()
{
while (LineNumber < LineCount && !LineIsEmpty(LineNumber))
    Down;

Down;
StartOfLine;
}
```

Add a newline after every closing angle bracket

```
// add a newline after each closing angle (>) character
// unless the angle already appears at end of line

macro AddNewlineAfterCloseAngle()
{
int line, i, j;
string str;

// loop on all lines in the file
for (line = 1; line <= LineCount(); line++)
    {
    // get the text of line 'line' into string 'str'
    GetLineText(line, str);

    // get the index of the closing angle
    j = strchr(str, '>');

    // if the character was found and was not at end-of-line...
    if (j != -1 && str[j+1] != '\0')
        {
        GotoLine(line);
        GotoColumn(1);

       // advance the cursor to the character
        while (ValueAtCursor() != '>')
```

```
            Right;

        // and past the character
        Right;

        // insert a newline
        Enter;

        // process this line again in case other tags exist
        line--;
        }
    }
}
```

Apply HTML markup to a simple text file

```
// apply HTML markup to a simple text file
// also converts double quote, ampersand, and
// angle brackets to HTML equivalents

macro ApplyHTMLMarkup()
{
int prevlen, len, i;
string str;
int numchanges;

// loop on all lines in the file
for (i = 1; i <= LineCount; i++)
    {
    // get the text of line 'i' into 'str'
    GetLineText(i, str);

    // reset the change counter
    numchanges = 0;

    // convert sensitive characters to HTML codes
    numchanges += ChangeString(str, "&",  "&amp;");
    numchanges += ChangeString(str, "<",  "&lt;");
    numchanges += ChangeString(str, ">",  "&gt;");
    numchanges += ChangeString(str, "\"", "&quot;");

    // if changes were made, replace the line's text
    if (numchanges > 0)
        PutLineText(i, str);
```

```
    }

// move to top of file
StartOfFile;
Down;

// loop on all lines in the file, starting on line 2
for (i = 2; i <= LineCount; i++)
    {
    // get the length of the previous line
    prevlen = LineLength(i-1);

    // get the length of this line
    len = LineLength(i);

    // if this line is empty, and the previous line isn't...
    // apply <br> markers to the end of the line
    if (len == 0 && prevlen != 0)
        {
        Up;
        EndOfLine;
        PutString("<br><br>");
        StartofLine;
        Down;
        }

    Down;          // move down to the next line
    }

StartOfFile;

PutString("<html>\n");
PutString("<head>\n");
PutString("<title></title>\n");
PutString("</head>\n\n");
PutString("<body>\n");

EndOfFile;
EndOfLine;
PutString("\n");
PutString("</body>\n");
PutString("</html>\n");

// place cursor between title and /title
GotoLine(3);
```

```
GotoColumn(8);
}
```

Display an ASCII chart in a new file

```
// ASCII chart example

macro ASCIIchart(void)
{
char i;

// open a new file
New;

// loop from space to 255 to show all chars
for (i = ' '; i <= 255; i++)
   printf("The ASCII value of '%c' is %d\n", i, i);
}
```

Convert comma-separated-value (CSV) data

```
// convert comma-separated-value (CSV) data on the current
// line so that each field is placed on its own line

macro ConvertCSV()
{
string str;
int numquotes, numcommas;

// get the count of quotes/commas on this line
numquotes = LineContains(linenumber, "\"");

numcommas = LineContains(linenumber, ",");

// if this appears to be CSV data...
if (numcommas+1 == numquotes / 2)
   {
   // get the text of the current line
   GetLineText(linenumber, str);

   // remove any empty data fields
   ChangeString(str, "\"\",", "");
```

```
    // convert "," to a newline
    ChangeString(str, "\",\"", "\n");

    // remove the first and last quotes
    ChangeString(str, "\"", "");

    // select the line
    GoToColumn(1);
    SelectToEndOfLine;

    // replace the selection
    PutString(str);
    }

// position for next line
Down;
StartOfLine;
}
```

Cut lines containing a user-defined string

```
// cut lines containing a user-defined string to the Windows clipboard

macro CutLinesContaining();
{
int line;
int len;
string str;
int numcut = 0;

// get the string from the user
len = GetString("Cut lines containing this string:", str);

if (len == 0)
    return;

// make the Windows clipboard the active clipboard
SetClipboard(0);

// clear the Windows clipboard
ClearClipboard(0);

// move cursor to start of file
StartOfFile();
```

```
// loop on all lines in the file
for (line = 1; line <= LineCount(); line++)
   {
   // does this line contain the string?
   if (LineContains(line, str))
      {
      GotoLine(line);
      CutAppend();
      numcut++;          // tally the cut
      line--;            // stay here for next line
      }
   }

// report the results
if (numcut == 1)
   message("Results", "1 line was cut to the Windows clipboard");
else
   message("Results", numcut,
           " lines were cut to the Windows clipboard");
}
```

Delete blank lines

```
// delete blank lines in the current file

macro DeleteBlankLines(void)
{
int i, len;

// start at the top of the file
StartOfFile;

// loop on all lines in the file
for (i = 1; i <= LineCount; i++)
   {
   // get the length of this line
   len = LineLength(i);

   // is this line empty?
   if (len == 0)
      {
      DeleteLine;     // delete this line
      i--;            // stay at this line #
```

```
      }
   else
      {
      Down;          // move down to the next line
      }
   }
}
```

Delete lines containing a user-defined string

```
// deletes lines containing a user-defined string

macro DeleteLinesContaining()
{
int line;
int len;
string str;
int deleted = 0;

// get the string from the user
len = GetString("Delete lines containing this string:", str);

if (len == 0)
   return;

// loop on all lines in the file
for (line = 1; line <= LineCount(); line++)
   {
   // does this line contain the string?
   if (LineContains(line, str))
      {
      DeleteLine(line);   // delete it
      deleted++;          // tally the deletion
      line--;             // stay here for next line
      }
   }

// report the results
if (deleted == 1)
   message("Results", "1 line was deleted");
else
   message("Results", deleted, " lines were deleted");
}
```

Delete lines NOT containing a user-defined string

```
// deletes lines NOT containing a user-defined string

macro DeleteLinesNotContaining()
{
int line;
int len;
string str;
int deleted = 0;

// get the string from the user
len = GetString("Delete lines that do NOT contain this string:", str);

if (len == 0)
   return;

// loop on all lines in the file
for (line = 1; line <= LineCount(); line++)
   {
   // does this line contain the string?
   if (!LineContains(line, str))
      {
      DeleteLine(line);   // delete it
      deleted++;          // tally the deletion
      line--;             // stay here for next line
      }
   }

// report the results
if (deleted == 1)
   message("Results", "1 line was deleted");
else
   message("Results", deleted, " lines were deleted");
}
```

Compute return on a deposited amount

```
// Compute result of amount left on deposit with continuous
// compounding.  Uses the formula: P = pe^rt

macro ComputeDeposit()
```

```
{
float amt, newamt, rate, years;
string str;

GetFloat("Enter the amount on deposit:", amt);

GetFloat("Enter the interest rate:", rate);

// if user entered 5, make it .05, for example
if (rate > 1.0)
   rate /= 100.0;

GetFloat("Enter the number of years on deposit:", years);

newamt = amt * pow(e, rate * years);

sprintf(str, "The amount with interest applied is: %.2f", newamt);
Message("Result", str);
}
```

Add blank lines after lines ending with !.?

```
// add a blank line after any line that ends with !.?

macro AddBlankLines(void)
{
char ch;
int i;

// loop on all lines in the file
for (i = 1; i <= LineCount; i++)
   {
   // make sure this line is not empty
   if (LineLength(i) > 1)
      {
      // move the cursor to this line
      GotoLine(i);

      // move to the end of the line
      EndOfLine;

      // backup off newline and onto last char
      Left;
```

```
        // get the value of char at the cursor
        ch = ValueAtCursor();

        // if it's a line ender, add Enter
        if (ch == '.' || ch == '?' || ch == '!')
            {
            EndOfLine;
            Enter;
            }
        }
    }
}
```

Double space and reformat

```
// double space and reformat the text on the clipboard
// prepares a web document for printing

macro DoubleSpaceAndReformat(void)
{
int i;
int numlines;

// save various editor settings
SaveSettings;

// open a new file and paste from clipboard
New;
Paste;

// set Text Width to 96
TextWidth(96);

// delete all blank lines
DeleteBlankLines;

// record the number of lines BEFORE we start adding lines
numlines = LineCount - 1;

// go to the top
StartOfFile;

// double space the file
for (i = 1; i <= numlines; i++)
```

```
    {
    Down;
    PutString("\n");
    }

// reformat the whole file
SelectAllText;
Reformat;

// remove any small indents that might be present
SelectAllText;
for (i = 1; i <= 12; i++)
Unindent;

// release the selection
Deselect;

// restore various editor settings
RestoreSettings;
}
```

Extract email addresses

```
// extract email addresses from all lines within the current file
// and append them to the end of the file

macro ExtractEmailAddresses()
{
int i, line, origlinecount;
int inword, isdelim;
int atsigns, dots;
int startword, endword;
string linetext, email;
char c;

// note the linecount before we start adding more lines
origlinecount = LineCount;

// loop on all lines in the file
for (line = 1; line <= origlinecount; line++)
    {
    // get the text of the whole line into the string 'linetext'
    GetLineText(line, linetext);

    // add a space to make end-of-line handling smoother
```

```
strcat(linetext, " ");

// loop on all characters in 'linetext'
for (inword = FALSE, i = 0; linetext[i] != 0; i++)
   {
   c = linetext[i];

   // set a flag if this character is one that delimits words
   if (isalnum(c) || (strchr("_@.-", c) != -1))
      isdelim = FALSE;
   else
      isdelim = TRUE;

   // decide whether this character starts a new word,
   // or ends an existing word
   if (inword && isdelim)
      {
      inword = FALSE;
      endword = i-1;

      // we've just left a word: see if it had both
      // the required characters
      if (atsigns == 1 && dots >= 1)
         {
         // get the linetext address into a string
         SubString(email, linetext, startword, endword-startword+1);

         // add it to the end of the file
         EndOfFile;
         EndOfLine;
         Enter;
         PutString(email);
         }
      }
   else if (!inword && !isdelim)
      {
      inword = TRUE;
      startword = i;
      atsigns = 0;
      dots = 0;
      }

   // tally whether or not we see the required chars while we're in a
   if (inword)
      {
      if (linetext[i] == '@')
```

```
            atsigns++;

        if (linetext[i] == '.')
            dots++;
        }
    }
  }
}
```

Hex to Decimal

```
// shows hex to decimal conversion technique

macro HexToDecimal()
{
string str;
int x = 0;
int i, val;
char ch;

GetString("Enter a hexadecimal string", str);

for (i = 0; str[i] != '\0'; i++)
   {
   ch = str[i];

   if (!isxdigit(ch))
      {
      message("Error", "Invalid character encountered: ", ch);
      return;
      }

   ch = toupper(ch);

   if (isalpha(ch))
      val = ch - 'A' + 10;
   else
      val = ch - '0';

   x = x * 16;
   x = x + val;
   }

message("Result", "The decimal value is ", x);
```

```
}
```

Obfuscate the selected text with HTML codes

```
// convert the word selected into its HTML coded format

// this can be used to convert phone numbers and email addresses
// in web pages to frustrate automated crawlers from harvesting
// your information for spam lists

macro Obfuscate()
{
int i;
string str;
string result;
string tmp;

if (!TextIsSelected)
   {
   Message("Error",
      "Please select a word before\nrunning the macro.\n");
   return;
   }

GetSelection(str);

// loop on all characters in 'str'
for (i = 0; str[i] != '\0'; i++)
   {
   sprintf(tmp, "&#%03d;", str[i]);
   strcat(result, tmp);
   }

PutSelection(result);
}
```

Display 24-hour time

```
// display the current time in 24-hour time format

macro Print24HourTime()
{
int h, m, s;
```

```
GetTime24(h, m, s);
printf("%d:%02d:%02d", h, m, s);
}
```

Reduce blank lines

```
// reduce multiple blank lines to one blank line

macro ReduceBlankLines(void)
{
int thislen, prevlen, i;

// position cursor to line 2
StartOfFile;
Down;

// loop on all lines in the file (starting with line 2)
for (i = 2; i <= LineCount; i++)
    {
    // get the length of the previous line
    prevlen = LineLength(i-1);

   // get the length of this line
   thislen = LineLength(i);

   // are both previous and this line empty?
   if (prevlen == 0 && thislen == 0)
      {
      DeleteLine;     // delete this line
      i--;            // stay at this line #
      }
   else
      {
      Down;           // move down to the next line
      }
   }
}
```

Reformat to an alternative text width

```
// Reformat the current paragraph to 70 characters, regardless
```

```
// of what the current Text Width setting is

macro ReformatAlternative()
{
SaveSettings;
TextWidth(70);
Reformat;
RestoreSettings;
}
```

Extract double quoted strings

```
// extract double quoted strings from the current file
// and append them at the bottom of the file

macro ExtractStrings()
{
string s, s1, s2, s3;
int i, j, k;
int found = 0;
int original_linecount = LineCount();

// loop on all lines in the current file
for (i = 1; i <= original_linecount; i++)
   {
   // does this line have two or more double quotes?
   while (LineContains(i, "\"") >= 2)
      {
      // tally number of strings found
      found++;

      // get the text of line 'i' into string 's'
     GetLineText(i, s);

      // get the offset of the first double quote
      j = strchr(s, '\"');

      // get the index of the second double quote
      for (k = j + 1; s[k] != '\"'; k++)
         ;

      // get the first portion into 's1'
      SubString(s1, s, 0, j);
```

```
        // get the second portion (the string) into 's2'
       SubString(s2, s, j, k-j+1);

        // get the third portion into 's3'
       SubString(s3, s, k+1, 2048);

        // build the new line and replace it
        s = s1;
        s += s3;
       PutLineText(i, s);

        // gather the strings at the bottom of the current file
       EndOfFile();
       EndOfLine();
       printf("\n%s", s2);
        }
    }

// report the results
if (found == 1)
    printf("\n\n%d string was found and removed\n", found);
else
    printf("\n\n%d strings were found and removed\n", found);
}
```

Reverse the text of each line

```
// reverse the text on every line in the file
// "abcdefg" becomes "gfedcba"

macro ReverseLineText()
{
int i, len, line;
string str;
char tmp;

// loop on all lines in the file
for (line = 1; line <= LineCount; line++)
    {
    len = linelength(line);

    // ignore lines too short/long
    if (len >= 2 && len < 2000)
        {
```

```
        // get the text of line 'line' into string 'str'
       GetLineText(line, str);

        // loop through half this line
       for (i = 0; i < len/2; i++)
           {
           // swap the characters...
           tmp = str[i];
           str[i] = str[len-1-i];
           str[len-1-i] = tmp;
           }

        // replace line with reversed line
        PutLineText(line, str);
        }
    }
}
```

Reverse names: Smith.Bob to Bob.Smith

```
// changes a list of "Smith.Bob" entries to "Bob.Smith"

macro ReverseNames()
{
int line, i;
string str;
string first, last;
string newstring;

// loop on all lines in the file
for (line = 1; line <= LineCount; line++)
    {
    // get the text of line 'line' into string 'str'
    GetLineText(line, str);

    // look for a '.' within 'str'
    i = strstr(str, ".");

    if (i != -1)
        {
        // 'last' gets 'i' chars from 'str' starting at index 0
        SubString(last,  str, 0,   i);

        // 'first' gets up to 100 chars from 'str' starting at index i+1
```

```
        SubString(first, str, i+1, 100);

        // build a new string from 'first' and 'last'
        sprintf(newstring, "%s.%s", first, last);

        // replace the text of the line
        PutLineText(line, newstring);
        }
    }
}
```

Truncate lines after a user-defined string

```
// truncate lines after a user-defined string

macro TruncateLineAfterString()
{
int j, line, len;
int truncated = 0;
string str, linestr, newstr;

// get the string from the user
len = GetString("Truncate lines after this string:", str);

// if the string is empty, quit
if (len == 0)
        return;

// loop on all lines in the file
for (line = 1; line <= LineCount; line++)
    {
    GetLineText(line, linestr);

    // does this line contain the string?
    if ((j = strstr(linestr, str)) != -1)
        {
        // create a new string without the trailing text
        SubString(newstr, linestr, 0, j+len);

        // replace this line with the new text
        PutLineText(line, newstr);

        // tally the truncation
        truncated++;
```

```
        }
    }

// report the results
if (truncated == 1)
    message("Results", "1 line was truncated");
else
    message("Results", truncated, " lines were truncated");
}
```

Truncate lines at a user-defined string

```
// truncate lines at a user-defined string

macro TruncateLineAtString()
{
int j, line, len;
int truncated = 0;
string str, linestr, newstr;

// get the string from the user
len = GetString("Truncate lines at this string:", str);

// if the string is empty, quit
if (len == 0)
            return;

// loop on all lines in the file
for (line = 1; line <= LineCount; line++)
    {
    GetLineText(line, linestr);

    // does this line contain the string?
    if ((j = strstr(linestr, str)) != -1)
        {
        // create a new string without the trailing text
       SubString(newstr, linestr, 0, j);

        // replace this line with the new text
       PutLineText(line, newstr);

        // tally the truncation
        truncated++;
        }
```

```
    }

// report the results
if (truncated == 1)
    message("Results", "1 line was truncated");
else
    message("Results", truncated, " lines were truncated");
}
```

Delete lines that begin with a user-defined string

```
// deletes lines that begin with a user-defined string

macro DeleteLinesThatBeginWith()
{
int line, len;
string str, linestr;
int deleted = 0;

// get the string from the user
len = GetString("Delete lines that begin with:", str);

if (len == 0)
    return;

// loop on all lines in the file
for (line = 1; line <= LineCount(); line++)
    {
            // get the text of line 'line' into 'linestr'
            GetLineText(line, linestr);
               // does this line start with 'str'?
            if (strncmp(linestr, str, len) == 0)
                {
                DeleteLine(line);   // delete it
                deleted++;          // tally the deletion
                line--;             // stay here for next line
                }
            }

// report the results
if (deleted == 1)
    message("Results", "1 line was deleted");
else
    message("Results", deleted, " lines were deleted");
```

```
}
```

Delete lines that end with a user-defined string

```
// deletes lines that end with a user-defined string

macro DeleteLinesThatEndWith()
{
int line, len, linelen;
string str, linestr, str2;
int deleted = 0;

// get the string from the user
len = GetString("Delete lines that end with:", str);

if (len == 0)
   return;

// loop on all lines in the file
for (line = 1; line <= LineCount(); line++)
   {
   // get the text of line 'line' into 'linestr'
   linelen = GetLineText(line, linestr);

   // if the line is too short, do nothing
   if (linelen < len)
      {
      ;
      }
   // does this line end with 'str' ?
   else
      {
      // isolate the tail of the line into a string
      SubString(str2, linestr, linelen - len, len);

      if (strcmp(str2, str) == 0)
         {
         DeleteLine(line);   // delete it
         deleted++;          // tally the deletion
         line--;             // stay here for next line
         }
      }
   }
```

```
// report the results
if (deleted == 1)
   message("Results", "1 line was deleted");
else
   message("Results", deleted, " lines were deleted");
}
```

Call MapQuest to show a map

```
// get an address from the user and call it up on MapQuest

macro CallMapQuest()
{
string city, state, address, country, url;
int zoom = 7;

// get information from the user
GetString("Enter street address:", address);
GetString("Enter city/town:",      city);
GetString("Enter state/province:", state);
GetString("Enter country:",        country);

// state level maps are better at zoom level 3
if (city == "")
   zoom = 3;

// country level maps are better at zoom level 1
if (state == "")
   zoom = 1;

// convert embedded spaces to plus signs
ChangeString(address, " ", "+");
ChangeString(city," ", "+");
ChangeString(state,   " ", "+");
ChangeString(country, " ", "+");

// build the URL that will be used
sprintf(url,
"http://www.mapquest.com/maps/map.adp?city=%s&state=%s&address=%s&c
ountry=%s&zoom=%d"    , city, state, address, country, zoom);

// send the URL to Windows so the default browser is run
OpenURL(url);
}
```

Convert British English punctuation to American English punctuation

```
// Convert British English punctuation to American English punctuation
// (in Britain, periods and commas are placed outside double quotes)

macro BritishPunctuation()
{
// notice that the double quote character must be escaped with a
// backslash when it appears within a string

// change ".  to  ."
ReplaceAll("\".",  ".\"");

// change ",  to  ,"
ReplaceAll("\"\,", ",\"");
}
```

4.7.5 Record Keys

Menu: Tools > Record Keys

Default Shortcut Key: Shift+F5

Macro function: none

Use the Record Keys command to begin recording a series of keystrokes for later playback with the Playback Keys command. This command provides a *'macro-by-example'* or *'on-the-fly'* keystroke recording facility. While recording is in process, all keystrokes entered from within the editor window will be stored. Mouse motion is not recorded *per se*, but if the mouse is used to select a command from the main menu, that command will be recorded just as though its shortcut key had been used instead. To stop recording, simply issue this command again: its name in the main menu changes to *Stop Recording* while recording is in process.

Mouse and/or keystroke interaction with popup dialogs is not stored in a key recording. If a confirmation dialog appears during keystroke recording, the response to that dialog is *not* recorded. The dialog will appear during playback, and will need to be dismissed manually at run-time.


If you need to pause in the middle of a keystroke recording, use the Pause Recording command. This will allow you to perform operations in the editor that will not appear in the key recording. Issue the command again to resume recording.


Command key assignments are not used in the files created by Save Key Recording,

so there's no chance that changing key assignments might disturb an existing key recording. Likewise, key recording files can be freely shared with other Boxer users without concern that playback might be influenced by differing key assignments on the target machine. However, depending on the nature of the key recording, default settings on the target machine (autoindent, typing wrap, etc) could influence how a key recording performs upon playback.

☞ There are several system-reserved key sequences that cannot be used within a keystroke recording. These key sequences all relate to moving among or closing document windows. The operating system traps these key sequences before they're ever seen by the application. Boxer provides its own key assignments for these commands, and they're available from the main menu as well, so it's not the *functionality* that need not be avoided. Rather, the *method of activating* these functions might need to be altered when making a recording. The key sequences to be avoided are the following: Ctrl+Tab, Shift+Ctrl+Tab, Ctrl+F4, Shift+Ctrl+F4, Ctrl+F6, Shift+Ctrl+F6.

☞ The Auto-Complete feature is disabled during keystroke recording. Since the entries that are presented in the pop-up Auto-Complete suggestion list are sensitive to the file being edited, and to text that has been previously typed, it's not safe to assume that the order of the list will be the same at playback time.

☞ If you need to automate a repetitive task in which logic must be applied to the recording, use Boxer's full-powered Macro language.

4.7.6 Pause Recording

Menu: Tools > Pause Recording

Default Shortcut Key: none

Macro function: none

Use the Pause Recording command to temporarily stop a keystroke recording that's already in process. Once paused, editor commands are not stored in the active recording. To resume recording, simply issue this command again: its name in the main menu changes to *Resume Recording* when recording is paused.

4.7.7 Playback Keys

Menu: Tools > Playback Keys

Default Shortcut Key: F5

Macro function: none

Issue the Playback Keys command to playback the most recent key recording. Execution begins immediately, so make sure that the text cursor is positioned as desired before proceeding.

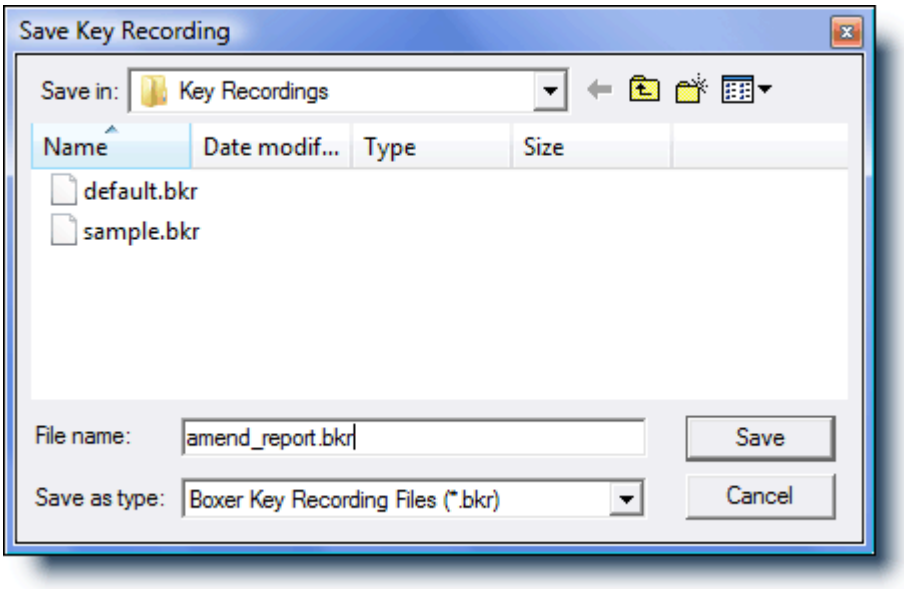
4.7.8 Save Key Recording

Menu: Tools > Save Key Recording

Default Shortcut Key: none

Macro function: none

Use the Save Key Recording command to save a key recording to a disk file. A dialog will appear that allows a name to be entered. By default, key recordings are stored in Boxer's 'Key Recordings' subdirectory.



Command key assignments are not used in the files created by Save Key Recording, so there's no chance that changing key assignments might disturb an existing key recording. Likewise, key recording files can be freely shared with other Boxer users without concern that playback might be influenced by differing key assignments on the target machine. However, depending on the nature of the key recording, default settings on the target machine (autoindent, typing wrap, etc) could influence how a key recording performs upon playback.

4.7.9 Load Key Recording

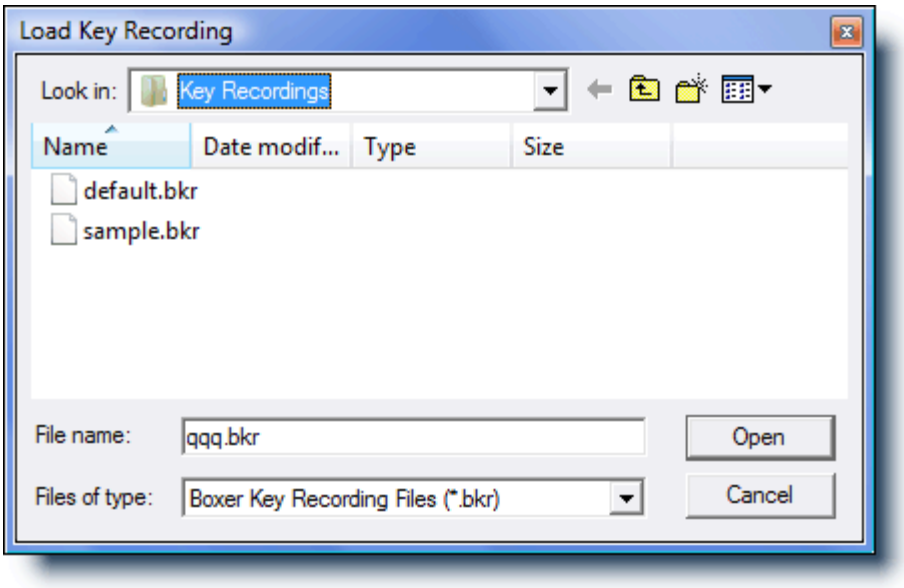
Menu: Tools > Load Key Recording

Default Shortcut Key: none

Macro function: none

The Load Key Recording command can be used to load an existing key recording from disk. A dialog will appear that allows a name to be selected. By default, key recordings

are loaded from Boxer's 'Key Recordings' subdirectory.



Once a key recording has been loaded, use the Playback Keys command to playback the recording.

4.7.10 Auto-Complete

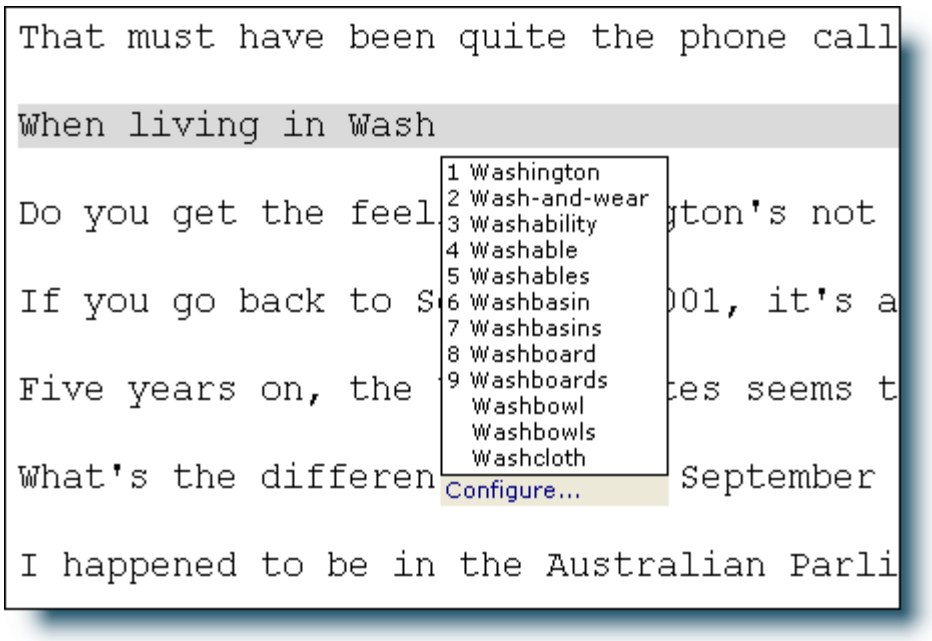
Menu: Tools > Auto-Complete

Default Shortcut Key: Ctrl+Space

Macro function: none

Boxer's Auto-Complete feature will display a popup list of matching words that can be used to complete a partially typed word at the text cursor. The popup list will appear after a (user-configurable) number of characters have been entered, and contains only those words that match the characters just typed. There is no need to interact with the list: if you prefer to keep typing, the list will simply disappear when it becomes irrelevant.

To select a word from the popup list, simply press the number displayed to its left, or use the down arrow to cursor into the list and press *Enter* at the desired entry.



The display (and recognition) of the hot numbers within the popup list is optional. It can be controlled on the Configure | Auto-Complete | Popup List dialog page.

If the suggestions within the popup list contain digits, the hot numbers feature will be automatically disabled. Otherwise, the entry of alphanumeric text strings would become confusing and error prone.

Discussion

Much of the utility of the Auto-Complete feature derives from the relevance of the words that are presented in the popup list.
Auto-Complete builds its list of matching words from four sources:

- user-defined phrases
- the existing text of the current document
- the reserved word list for the current file type (when applicable)
- an external, user-editable dictionary of 37,000 long words

User-defined phrases are those words and phrases that have been pre-defined on the Configure Auto-Complete - User-Defined dialog page. These might include common text strings that you use in your work, your mailing address, email address, etc. These phrases are given highest priority and will appear first in the completion list.
User-defined phrases can also be set to expand 1) as soon as they're typed (bte could auto-expand to 'Boxer Text Editor'), 2) only when a delimiter is typed, or 3) only when the Trigger Key is pressed. See Configure Auto-Complete - User-Defined for full details.

Auto-Complete also analyzes the content of the current file to find potential matches for the completion list. This means that words and phrases that are particular to your document are automatically suggested as matching words. If you're editing a document that contains the term 'diethylthiocarbamate', that word will appear in the

completion list, even though it doesn't appear in the dictionary proper. Auto-Complete also intelligently harvests phrases from program source code, making the feature useful for programmers as well.

When editing a file for which Syntax Highlighting has been defined, the reserved words for that language will be used as completion words.

Finally, Auto-Complete uses a large dictionary of English* words from which all shorter words have been removed. This dictionary is maintained in simple, uncompressed ASCII text format, and is therefore user-editable. The dictionary can be viewed and/or edited from the Configure Auto-Complete - Dictionary dialog page.

☞ * Dictionaries for other languages are not available at this time, but if you can locate a large word list for the language of interest, you can use that list to replace the `AC_Words.txt` file provided with Boxer. The Sort Lines command has an option to sort lines by line length, making it easy to locate and remove smaller words from the list. If you assemble your list from multiple sources, use the Delete Duplicate Lines to remove duplicates. The final word list should contain one word/phrase per line, and be ASCII-sorted, case insensitive.

Customization

Auto-Complete is the type of feature that some users will have very strong feelings about. Users are sure to have different ideas about *when* the popup list should appear, *where* it should appear, *whether* it should appear at all, *how many* entries it should have, *which* words should appear in the list, etc. Auto-Complete has an **extensive** collection of configuration options to control every aspect of its operation. The following help topics cover the configuration of the Auto-Complete feature:

General Settings
Popup List Settings
User-Defined Phrases
Harvested Words
Dictionary Words
Excluded Words

If the default Auto-Complete settings don't feel perfect to you, you are encouraged to spend a few minutes experimenting with the various options to make sure it feels just right.

Manual Operation

Some users may find the popup list distracting, and opt to disable it. When Auto-Complete is configured not to display a popup list, the *Trigger Key* can be used to complete the partially typed word at the text cursor. Pressing the Trigger Key repeatedly cycles through the available matches. By default, the Trigger Key for Auto-Compete is *Ctrl+Space*.

The popup list can be displayed at any time by using the Auto-Complete List command. This command is useful when the popup list has been disabled, or when you want to force the list to appear in a situation where it would not naturally appear (for example, when you haven't typed enough characters for it to appear).

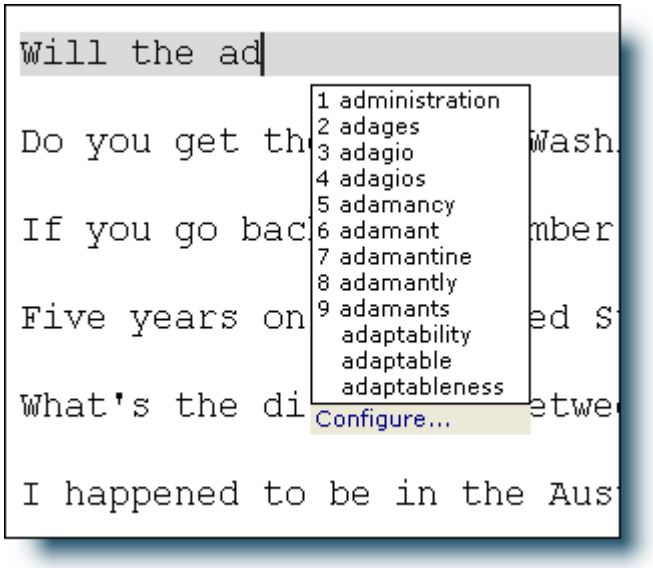
4.7.11 Auto-Complete List

Menu: Tools > Auto-Complete List

Default Shortcut Key: Ctrl+Alt+Down

Macro function: none

The Auto-Complete List command can be used to force the popup list of matching words to appear in situations when it wouldn't appear naturally on its own. One such case is when an insufficient number of characters has been typed to cause the list to appear. Another case is when the popup list has been configured not to appear at all.



For more complete information about Auto-Complete, see the Auto-Complete topic.

4.7.12 Command Multiplier

Menu: Tools > Command Multiplier

Default Shortcut Key: Alt+Y

Macro function: none (Boxer's macro language provides far more powerful methods to multiply the execution of commands)

The Command Multiplier can be used to multiply the execution of a keystroke or command key sequence. A popup box appears to retrieve the multiplier to be used. After clicking OK, Boxer awaits the next keystroke or command key sequence. Once issued it will be performed repeatedly, according to the value entered.

This command might be used to multiply the execution of an insertable character so as

to create a divider bar containing a known number of characters. Or it might be used with the Delete Current Line command to quickly delete 100 lines.

To simply repeat the last command issued, one or more times, use the Repeat Last Command command.

4.7.13 Repeat Last Command

Menu: Tools > Repeat Last Command

Default Shortcut Key: F10

Macro function: none

Repeat Last Command can be used to repeat the most recently issued command. This command is especially convenient when the command last issued does not have a shortcut key, and must be executed from the pull-down menus. Repeat Last Command is assigned to the *F10* key, by default, to ensure it can be easily executed.

Repeat Last Command cannot be used to repeat cursor movement commands such as Up, Down, Left, Right, etc.

Repeat Last Command can also be used to repeat the insertion of single characters. When a character is not readily typed from the keyboard (think high-ASCII characters), Repeat Last Command can be a real time saver.

4.7.14 Format XML / XHTML

Menu: Tools > Format XML / XHTML

Default Shortcut Key: none

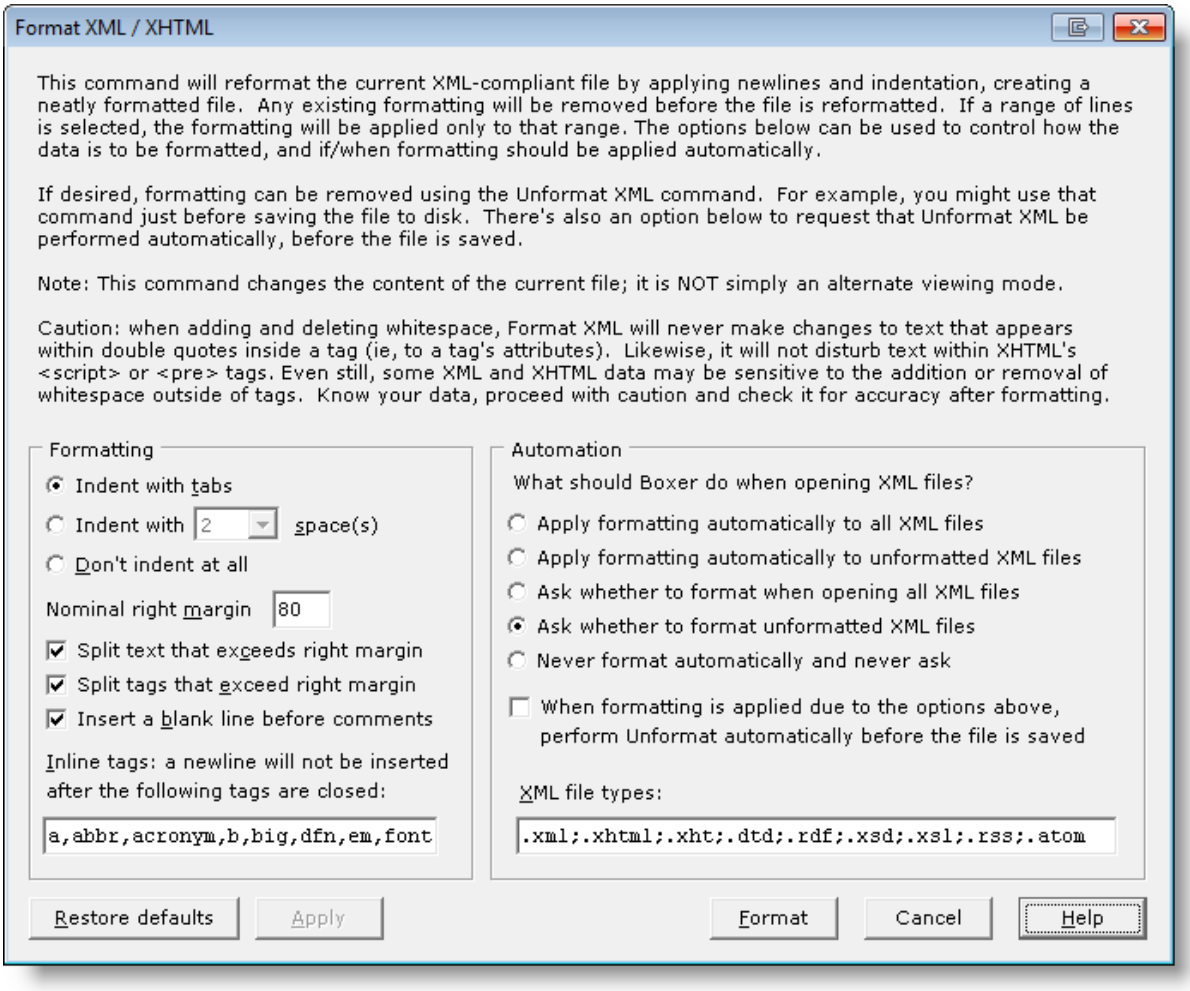
Macro function: FormatXML()

The Format XML / XHTML command can be used to apply formatting to XML-compliant files. If you've ever worked with XML files, you may have noticed that these files often lack line enders and indenting. When these files are opened in Boxer, the text appears as a single long line, and flows off-screen and out of sight at the right edge of the window. The absence of formatting presumably provides some efficiency to the software programs that process these files, but does so at the expense of human readability. The Format XML / XHTML command can neatly format these files with line enders and proper indentation.

The Unformat XML / XHTML command can be used to remove formatting.

By default, the formatting operation is applied to the whole file. If a range of lines is selected, formatting will be performed on the range of lines selected.

A variety of options are provided on the Format XML dialog to control the formatting process:



Formatting

Indentation
Three options are provided: indent with tabs, indent with *n* spaces, don't indent at all. If tabs are used, the display value of a tab is governed by the Tab Display Size command. A large indent value can make it easier to see the structure of the data file. But if a document contains deeply nested data blocks, a more modest indent value may need to be used to preserve space.

Right Margin
This option controls the nominal margin at which lines will be split in order to maintain the width of the document. Please note that double-quoted strings will *not* be split in an attempt to stay within the margin. Lines will be split only at legal break points between or within tags.

Split text that exceed right margin
This options controls whether or not text data will be split/wrapped in order to maintain the right margin.

Split tags that exceed right margin
This options controls whether or not tags (with spaces) will be split/wrapped in order to maintain the right margin.

Insert a blank line before comments
This option can be used to force an empty line to appear before a comment line, or a group of comment lines.

Inline tags
This option can be used to list those tags which will be treated as inline tags. When an inline tags is closed, a newline is not added to the output. Inline tags are typically those tags that might be used to apply formatting to a word or phrases, and are not those tags that begin a block of data which will include other tags.

Automation

A variety of options are offered to control when and whether XML formatting should be applied automatically.

Apply formatting automatically to all XML files
If this option is checked, formatting will be applied to all XML files as they are opened, whether formatted or not, without asking.

Apply formatting automatically to unformatted XML files
If this option is checked, formatting will be applied to unformatted XML files as they are opened, without asking. Formatted XML files with be opened without modification.

Ask whether to format when opening all XML files
If this option is checked, Boxer will ask whether formatting should be applied each time an XML file of any kind is opened.

Ask whether to format unformatted XML files
If this option is checked, Boxer will ask whether formatting should be applied each time an unformatted XML file is opened. Formatted XML files with be opened without modification.

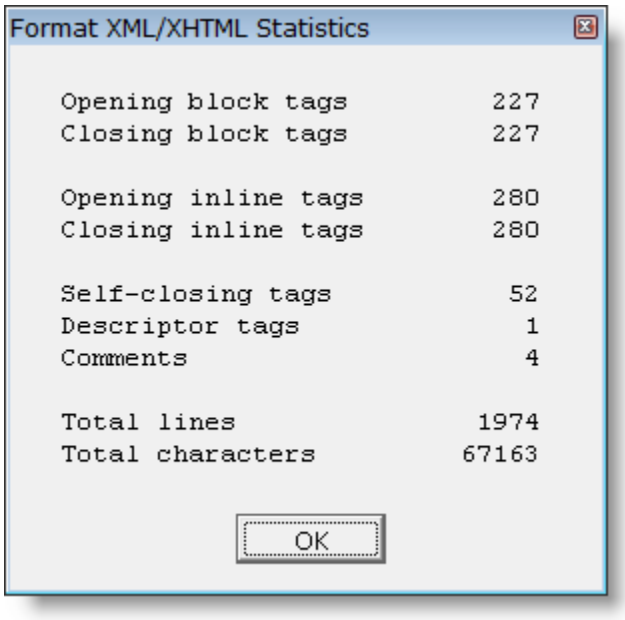
Never format automatically and never ask
If this option is checked, Boxer will never ask about formatting XML files, and will never format them. In this case, the Format XML could be invoked manually in order to initiate formatting.

When formatting is applied due to the options above, perform Unformat automatically before the file is saved
Use this option to ensure that any formatting that is applied due to the automation options is removed before a file is saved. Please note: if the Format XML dialog is summoned manually to add formatting, you'll need to manually remove formatting with Unformat XML, if desired.

XML file types
This option lists the file extensions of those files which are consider XML files, for purpose of automated formatting.

Statistics

When formatting is applied manually (ie, not via automation), a statistics dialog is displayed upon completion:



The dialog can be helpful in locating mismatched or unclosed tags. In particular, the task of converting HTML files to XHTML (XML-compliant HTML) can be aided considerably by using the Format XML command. By using the statistics dialog, and watching for inconsistent indenting, you'll be able to locate which tags are preventing your HTML file from being XML-compliant. A proper XHTML file should start and end at zero indent. If the indent level grows over the course of the document, this is probably an indication that one or more self-closing tags have not been closed. For example, `<br>` needs to be changed to `<br />`.

☞ Text found within `<script>` and `</script>` tags (or within `<?` and `?>`) will not be formatted by this command. Embedded scripting code may be sensitive to indentation and wrapping, so Format XML will not process such text during its operation.

4.7.15 Unformat XML / XHTML

Menu: Tools > Unformat XML / XHTML

Default Shortcut Key: none

Macro function: UnformatXML()

The Unformat XML / XHTML command can be used to remove formatting (newline and indentation) from an XML or XHTML file. This command can be used to remove formatting that was added by the Format XML / XHTML command, or to remove formatting from a file from some other source. After formatting is removed, the file will reside on one long line (subject to Boxer's limitation on line length).

By default, the unformat operation is applied to the whole file. If a range of lines is selected, unformat will be performed on the range of lines selected.

Please see the Format XML / XHTML command for a lengthy discussion of this topic.

4.7.16 Spell Checker

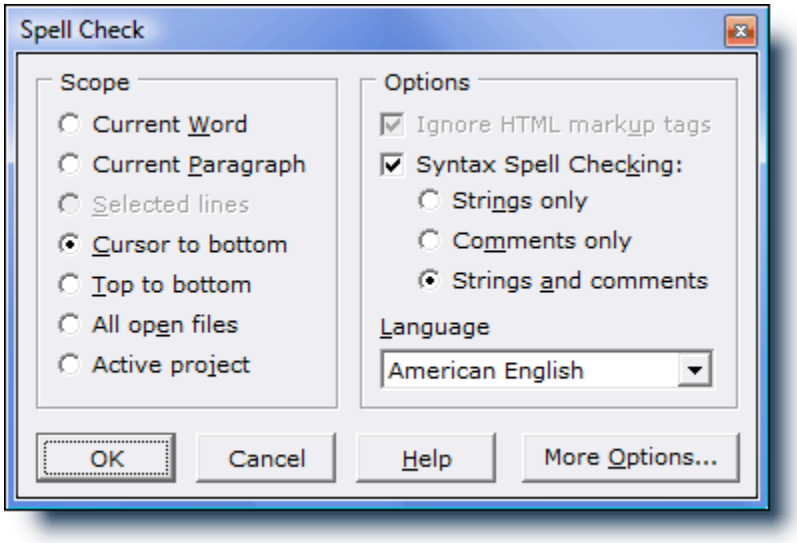
Menu: Tools > Spell Checker

Default Shortcut Key: F7

Macro function: SpellChecker()

The Spell Checker command provides access to the built-in spell checking facility. Boxer's spell checker provides several options to control the scope of the spell check operation, and some unique options which permit spell checking to be performed within program source code files.

For information about Boxer's on-the-fly spell checking feature, which checks text as you type, see the Active Spell Checking command.



Scope

Current Word
Use this option if only the word at the text cursor is to be checked.

Current Paragraph
Use this option to spell check the current paragraph only.

Selected lines
Use this option to constrain the spell check to the selected text. When text is not selected, this option will be disabled.

Cursor to bottom
Use this option to spell check from the cursor to the end of file.

Top to bottom
Use this option to spell check the entire file.

All open files
Use this option to spell check all open files.

Active project
Use this option to limit the scope of the spell check to those files within the active project.

Options

Ignore HTML markup tags
When an HTML file is being edited, this option can be used to tell the Spell Checker to ignore HTML markup tags, thereby avoiding the false errors which would occur if the entire file were checked.

Syntax Spell Checking
When editing a file for which Boxer has Syntax Highlighting information, this option can be used to tell the Spell Checker to check only that text which matches a designated syntax. This makes it possible to spell check program code without getting false hits for reserved words, variable names, and other text which naturally cannot appear within a dictionary. When this box is checked, the options below become available for selection.

Strings only
Use this option to spell check only that text which appears in String context. Boxer uses its Syntax Highlighting information to determine the syntax of the text being checked.

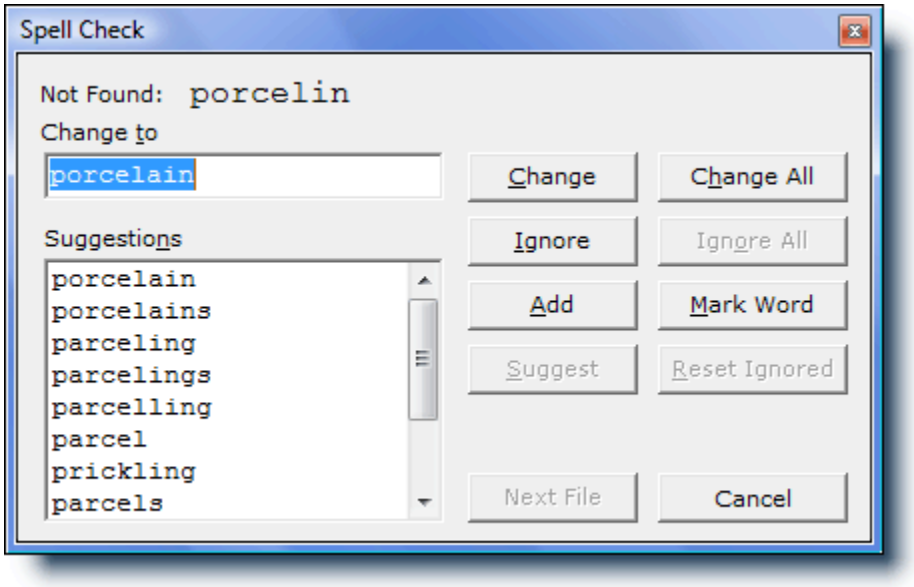
Comments only
Use this option to spell check only that text which appears in Comment context. Both block comment text and end-of-line comment text will be checked. Boxer uses its Syntax Highlighting information to determine the syntax of the text being checked.

Strings and Comments
Use this option to spell check only that text which appears in either String or Comment context. Both block comment text and end-of-line comment text will be checked. Boxer uses its Syntax Highlighting information to determine the syntax of the text being checked.

Language
Use this option to select the dictionary to be used by the Spell Checker. Dictionaries are available for the following languages: American English, British English, Dutch, French, German, Italian, Czech and Spanish. Three specialty dictionaries are also available: Legal, Medical and 'Moby', an extra large American English dictionary. If Boxer was not supplied with the dictionary you prefer to use, please visit our website at www.boxersoftware.com to obtain our other dictionaries. The dictionaries are located in the Downloads area.

More Options...
The *More Options* button provides a quick way to summon the Configure | Preferences | Other dialog page, which contains additional Spell Checker options.

After specifying the scope and other options, the Spell Check operation begins. When a word is encountered which is not found in the dictionary, a popup dialog box is presented:



The position of the popup box is selected so as not to overlap the offending word within the editor window. The word that was not found is displayed in bold, and is also highlighted in the editor window to show its context. Several options are available:

Change to
The *Change to* edit box displays a suggested word which the offending word might be replaced with. The text in this box can be edited as may be needed.

☞ To ensure that special characters are displayed in the *Change to* box as they will appear when inserted into the editor, the *Change to* box uses the same font as is used in the editor itself.

Suggestions
The *Suggestions* listbox displays a list of other words which might be used to replace the offending word. Click on a word to select it and move it into the *Change to* edit box.

Change
Use the *Change* button to replace the offending word with the word in the *Change to* box.

Change All
Use the *Change All* button to replace the offending word with the word in the *Change to* box, and to indicate that all future occurrences of the word should also be changed.

Ignore
Use the *Ignore* button to skip the offending word. Future occurrences of the word will be presented when they occur.

Ignore All
Use the *Ignore All* button to skip the offending word, and to indicate that all future occurrences should also be ignored.

Add
Use the *Add* button to add the offending word to the dictionary. Words which are added to the dictionary are saved within the file `userdict.txt` in Boxer's *data folder*. This file can also be edited within Boxer to add other words, or to remove words which may have been added mistakenly.


Words which are added to the user dictionary will be accepted as correctly spelled words in any case configuration in which they may occur. For example, if the word `ebay` is added to the dictionary, it will be accepted in any of the following forms: `eBay`, `ebAy`, and `ebaY`. This liberal processing was necessary because the third-party dictionary that Boxer uses is not processed in a case sensitive manner. Before this handling was put in place, the word `eBay` would always be reported as misspelled, even when `eBay` (or any variant) had been added to the user dictionary.

Mark Word
Use the *Mark Word* button to surround the offending word with pound signs (###). This makes it easy to locate the word later on to make manual adjustments. When the Spell Check operation is complete, the cursor will be placed on the first marked word.

Suggest
Once a change has been made to the word in the *Change to* box, the *Suggest* button can be used to fill the *Suggestions* listbox with other words which may be related to the word.

Reset Ignored
Use the *Reset Ignored* button to clear the list of ignored words which has been built during the current edit session.

Next File

Use the *Next File* button to skip the rest of the current file and move to the next file to be checked.

By default, Boxer is configured to use the American English dictionary. Dictionaries are also available for British English, Dutch, French, German, Italian, Czech and Spanish. Three specialty dictionaries are also available: Legal, Medical and 'Moby', an extra large American English dictionary. The active dictionary can be set on the dialog that appears when the Spell Checker is first run. If Boxer was not supplied with the dictionary you prefer to use, please visit our website at www.boxersoftware.com to obtain our other dictionaries. The dictionaries are located in the Downloads area.

Several options which can be used to further control the Spell Checking process are available on the Configure | Preferences | Other options page.

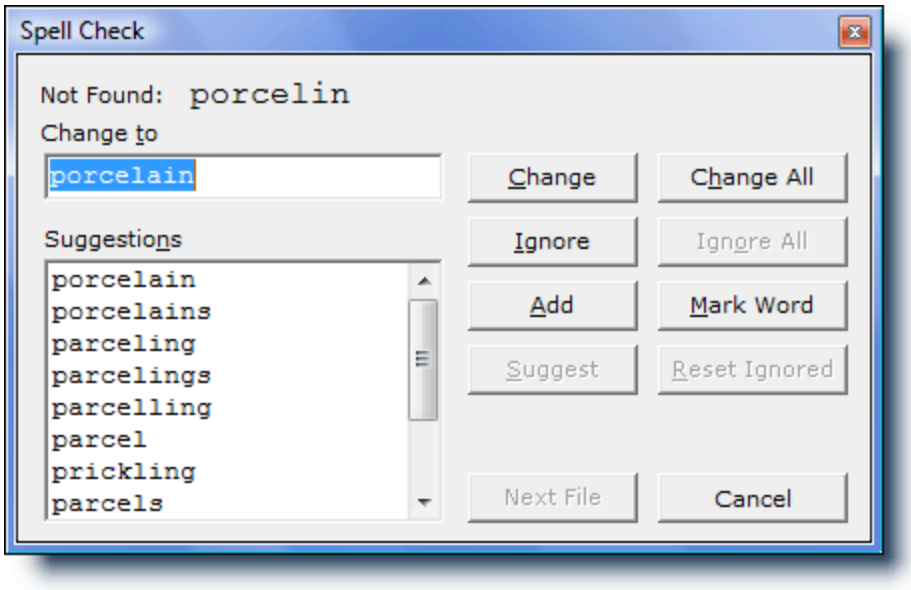
4.7.17 Check Word

Menu: Tools > Check Word

Default Shortcut Key: Shift+F7

Macro function: CheckWord()

Use the Check Word command to check the spelling of the word at the text cursor. If the word is spelled correctly, a message will appear to confirm this fact. If the word is spelled incorrectly, a dialog will appear providing options to make a correction:



The Check Word command can also be accessed by right-clicking on a suspect word and selecting the command from the context menu.

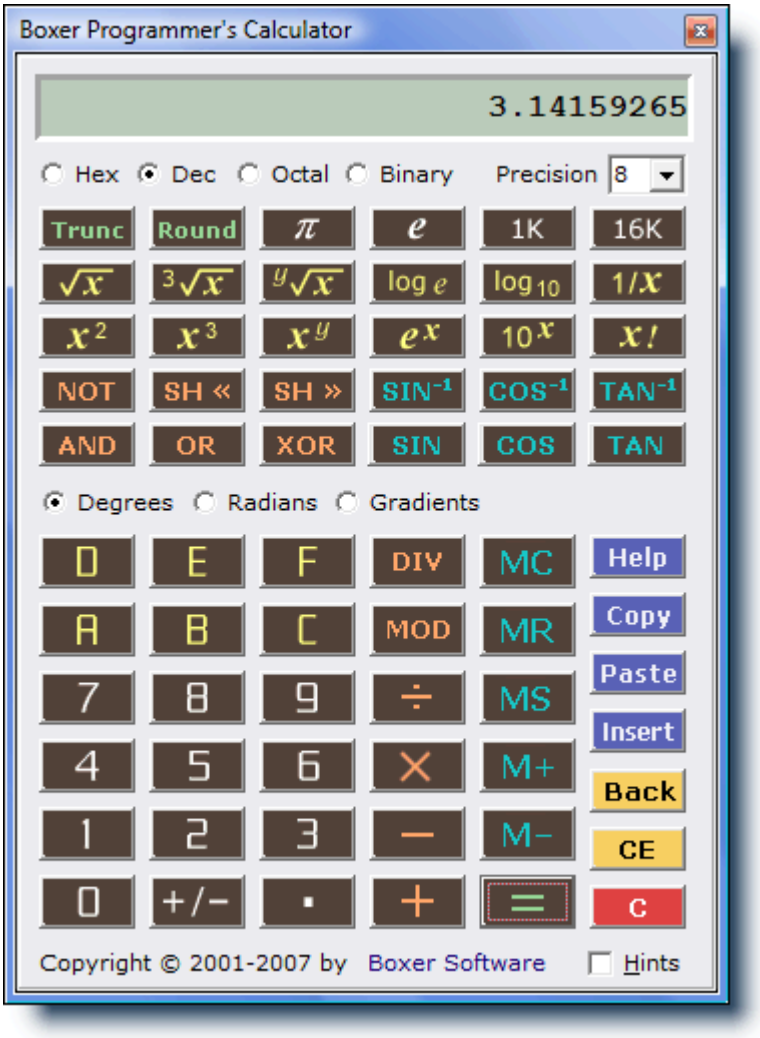
4.7.18 Calculator

Menu: Tools > Calculator

Default Shortcut Key: F11

Macro function: Calculator()

The Calculator command provides access to Boxer's multi-function, multi-base calculator:



Boxer's Calculator works just like a conventional calculator, and it has all the scientific and trigonometric functions one would expect to find on a full-featured calculator. Values can be entered by clicking keys with the mouse, or by using the keyboard. A list of shortcut keys can be found below.

If a numeric value appears beneath the text cursor when the Calculator is summoned, that value will be placed into the calculator display automatically. The Calculator is also

able to interact with the clipboard. The *Copy* button can be used to copy the value in the Calculator's display to the Windows clipboard. The *Paste* button can be used to paste a value from the clipboard into the Calculator's display. The *Insert* button will insert the value in Calculator display at the current text cursor location.

Selecting the *Hints* checkbox reveals a small panel at the bottom of the Calculator that displays information about the key below the mouse cursor.

Calculator Shortcut Keys

Degrees Mode	F2
Radians Mode	F3
Grads Mode	F4
Hexadecimal Mode	F5
Decimal Mode	F6
Octal Mode	F7
Binary Mode	F8
Precision	F10
Pi	P
Euler's Constant	E
1 Kilobyte	K
16 Kilobytes	Ctrl+K
Square Root	Q
Log - natural	N
Log - base 10	L
Reciprocal	R
Square	@
Cube	#
Y-th Power	Y
e to the X	X
Factorial	!
NOT	~
AND	&
OR	|
XOR	^
Shift Left	<
Shift Right	>
Sine	S
Cosine	O
Tangent	T
Add	+
Subtract	-
Multiply	*
Divide	/
Modulus	%
Memory Add	Ctrl+P
Memory Subtract	Ctrl+S
Memory Store	Ctrl+M
Memory Recall	Ctrl+R

Memory Clear	Ctrl+L
Copy to Clipboard	Ctrl+C
Paste to Display	Ctrl+V
Insert into File	Ctrl+I
Clear	Esc
Clear	C (unless in hex mode)
Clear Entry	Del
Back	Backspace
Help	F1
Close	Alt+F4

☞ The Calculator uses 64-bit arithmetic so that *very large* values can be entered and computed. Special thanks are due to Jonas Hammarberg for his help in this area.

☞ The Calculator uses the Algebraic Operating System (AOS), not Reverse Polish Notation (RPN).

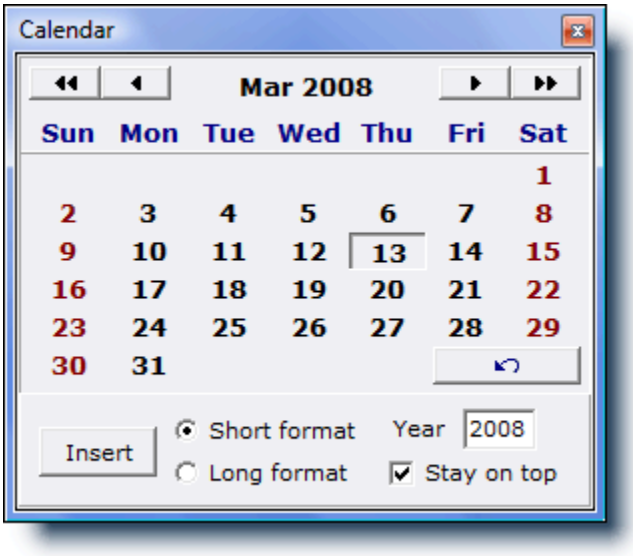
4.7.19 Calendar

Menu: Tools > Calendar

Default Shortcut Key: none

Macro function: Calendar()

The Calendar command provides access to Boxer's popup calendar:



When first summoned, the Calendar displays the current month and year and highlights

the current date. The arrow buttons at the top of the display can be used to move forward or backward, by months or by years. The button with the curved arrow can be used to return the display to the current month and year.

Clicking on any date within the display highlights that date. The *Insert* button can be used to insert a text string describing the highlighted date at the text cursor location. The *Short Format* and *Long Format* options control the format that will be used. The short and long formats used to display the date are in accordance with the regional settings for date display as defined on your computer. To change these settings, see Start Menu | Settings | Control Panel | Regional Settings | Date.

The Calendar recognizes various characters to speed movement from date to date:

```
Y = first day of year
R = last  day of year
M = first day of month
H = last  day of month
T = Today
```

The following keys are also recognized:

```
Space        = +1 month
-            = -1 day
+            = +1 day
Shift+Left   = -1 month
Shift+Right  = +1 month
Ctrl+Left    = -1 year
Ctrl+Right   = +1 year
```

The day on which a calendar week starts can be configured on the Configure | Preferences | Other options page. The option is titled: *Calendar week starts on*.

If you prefer that the Calendar always remain on top of other windows, the *Stay on top* option can be used. The Calendar is a *non-modal* window, which allows it to remain on-screen after *focus* has been returned to another editing window.


If the Calendar is left on-screen when Boxer is closed, it will be automatically reopened if the edit session is later restored.

4.7.20 User Tools

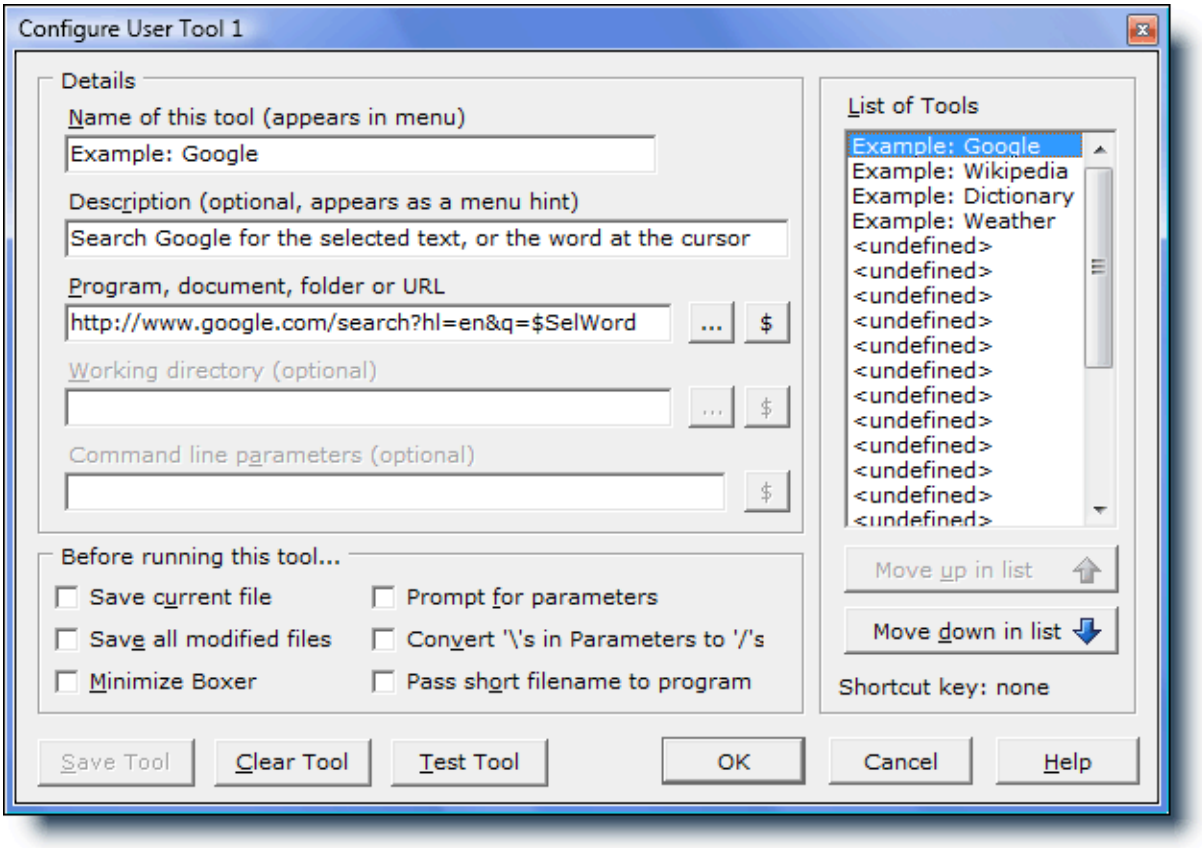
Menu: Configure > User Tools

Default Shortcut Key: none

Macro function: ConfigureUserTools()

The Configure User Tools command is used to define and configure up to 24 external programs which can be run from the User Tools submenu on the Tools menu. A variety of *Tools Macros* is available which makes it possible to control the information which is passed to the program being run.

A common use for a User Tool would be to send the name of the current file to an external program which processes the file in some way. Examples of such programs are assemblers, compilers, grammar checkers, parsers, etc.



Use of the Configure User Tools dialog box is described below:

Details

Name of this tool
Use this edit box to supply the name for the User Tool being defined. The name supplied will appear in the User Tools submenu when definition is complete. Up to 20 characters can be used.

Description
Use this edit box to supply an optional description for the tool being defined. This description will appear as a menu hint for the Tools | User Tools menu entry that corresponds to this tool.

Program, document, folder or URL
Use this edit box to supply the full filepath of the program which is to be run. The button with the ellipsis (. . .) can be used to browse for and select the desired program. If the program selected has an associated icon, it will be displayed to the right of the *Name* edit box.

Tools Macros can be placed into the *Program* field by clicking on the '$' button. For example, you might use the $SelWord directive to pass the word at the cursor--or a short text selection--to a web-based resource that performs a search for that term. The URL:

```
http://www.google.com/search?hl=en&q=$SelWord
```

would cause the word at the cursor to be sent to Google for search results.

The $Sel and $SelWord directives are processed specially when used in a URL: any embedded spaces that might result from expansion are automatically converted to plus signs (+) to create a web-friendly URL.

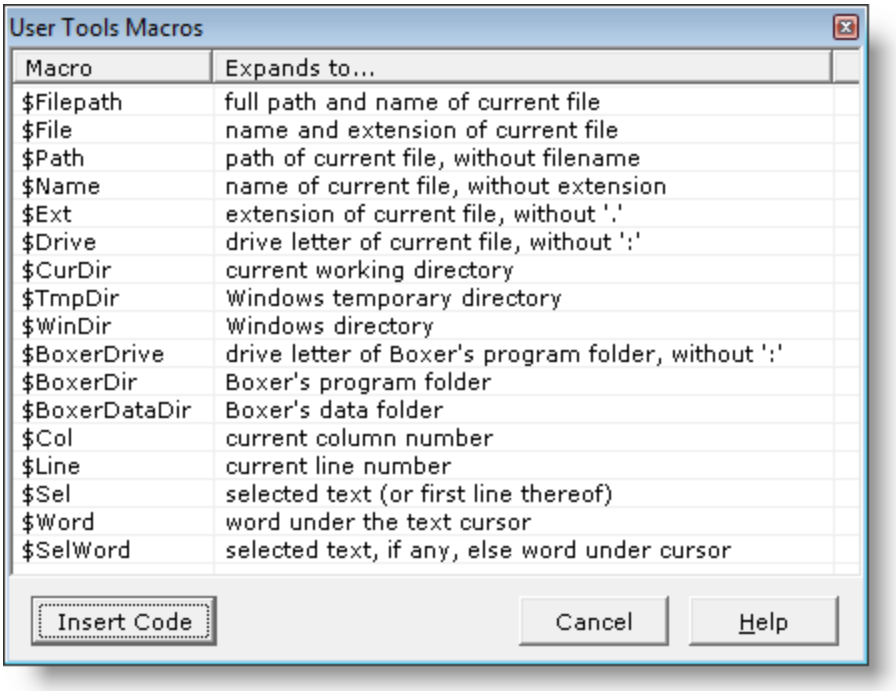
Working Directory
Use this edit box to supply the working directory for the program being defined. The working directory will become the current directory for the program being run. The button with the ellipsis (...) can be used to browse for and select the working directory.

The $Path Tools Macro (among others) can also be used in the *Working Directory* field as a means of specifying the working directory.

Command Line Parameters
The *Command Line Parameters* edit box is used to supply command line parameters to the program being defined. These parameters might be option flags required by the program or any other such information.

Boxer will recognize several different *Tools Macros* in the *Command Line Parameters* edit box to pass information to the defined program. When a macro appears in the *Command Line Parameters* field, it will be expanded to its equivalent text at the time the User Tool is run.

Clicking the ellipsis (...) button to the right of the edit box presents the *Tools Macros* list:



Macros are available to pass the current filepath, filename, extension and other portions of the filename. The line and column number can also be passed, as can the word beneath the text cursor. Selected text (or the first line of a multi-line selection) can also be passed. If needed, two or more macros can be placed in the *Command Line Parameters* edit box.

☞ If it is anticipated that the file and/or path being passed to the User Tool might contain an embedded space, be sure to enclose the $Filepath directive in double quotes to ensure proper handling.

Before running this program...

Save current file
Use this option to save any changes in the current file before running the defined User Tool. Be sure to use this option when defining a User Tool which will operate on the current file, so the program has access to the most recent changes you've made.

Save all modified files
Use this option to save any changes within all edited files before running the defined User Tool.

Minimize Boxer
Use this option to request that Boxer be minimized to the task bar while the User Tool is running.

Prompt for parameters
Use this option to request that Boxer prompt for parameters before running the defined program. This option is useful when running a program whose command line parameter(s) must be determined according to other conditions and cannot be specified

programmatically.

Convert '\'s in Parameters to '/'s
Use this option to request that any backslashes (`\`) within the *Command Line Parameters* edit box be converted to forward slashes (`/`) before running the defined program.

Pass short filename to program
Use this option to request that any *Tools Macros* used in the *Command Line Parameters* edit box be converted to *short filenames* before running the defined program. This option is needed when defining a User Tool which passes a filename to a DOS program, or to a 16-bit Windows program, since these programs are typically unable to process *long filenames*.

Buttons

Save Tool button
Use the *Save Tool* button to save the current tool definition. Note that the current tool will also be saved automatically when moving to a new tool in the *Tools* list.

Clear Tool button
Use the *Clear Tool* button to clear the definition for the current tool. No confirmation is required before the tool is erased.

Test Tool Button
Use the *Test Tool* button to simulate running the current tool, without actually executing the defined program or resource. A dialog will appear showing the various fields, after the expansion of any *Tools Macros* and other requested conversions have been made.

Move up in list / Move down in list
Use these buttons to change the order in which tools appear in the User Tools submenu on the Tools menu. Clicking on a button moves the currently selected User Tool up or down in the list. Moving a User Tool up or down in the list does not cause a change in the shortcut key assignments, if any are in use. For example: a shortcut key assigned to User Tool 2 remains assigned to the second tool in the list and does not travel with a tool which is moved through that position.

Tips and Notes

The method by which Boxer runs a User Tool program makes it possible to define User Tools which are mapped to documents, rather than programs. For example, if the 'program' to be run is defined to be an HTML document, then your Internet browser will be launched to display that file. If a `.DOC` file is defined as the 'program' to be run, then Microsoft Word will be launched to display the document. The browse button will present a file selection dialog box which defaults to showing `.EXE` and `.COM` files, so in order to locate documents it will need to be changed to show files of all types.

The method described in the above Tip can also be used to create User Tools which map to your favorite directories. If a directory name is defined as the 'program' to

be run, then Explorer will be launched with that directory in its view. The browse button cannot be used to select a directory name, so in order to define a User Tool in this way, the directory name will need to be typed manually into the *Program* edit box.

Some DOS programs issue an on-screen report but do not pause for user interaction or confirmation before terminating. When such programs are run as User Tools, they will execute and terminate so quickly that their results cannot be studied. You can remedy this behavior by making a change to the Properties of the program being executed. Locate the program in Explorer, right-click on its icon, and select Properties. Click the *Program* tab and uncheck the option titled *Close on exit*. Click *OK* to save the change. Thereafter, when the program is run, its window will not close until its close button is clicked.

By default, Boxer is configured to present a warning message when a file it is editing is changed by another program or process. This capability is especially useful when running User Tools since it confirms that a change was made and provides the opportunity to Reload the file from disk to get the latest copy. If you will be running a User Tool which operates on the current file, this option should be kept in force. The option is located on the Configure | Preferences | Messages option page, and is titled *Warn when an edited file is changed by another program.*

If you are a user of one of the JP Software command processors and wish to specify a `.BTM` file as a User Tool, you may need to make a system configuration change before doing so. To see if a change is required, create a `.BTM` file which performs some passive operation (such as `DIR`), and try to execute it by double clicking from within Explorer. If the file executes properly, then the Windows shell is aware that `.BTM` files can be executed, and no changes are needed. If the file does not run, then a change will be needed before a `.BTM` file can be run as a program in one of Boxer's User Tools. JP Software has documented this configuration procedure in an information file on their website. At the time of writing, this file could be found at: www.jpsoft.com/help/index.htm?deskobjs.htm If the file is not found there, look in the *Support* section at www.jpsoft.com.

Here's a tip for users of JP Software's 4DOS and/or 4NT command processors: If you would like to direct the error output from a DOS program to the Windows clipboard, you can make use of the `clip:` logical device to achieve this. At the end of any *Command Line Parameters* which might be defined for a given User Tool, add the following: `>&>! clip:` This directive causes the *standard error* stream to be placed on the Windows clipboard, overwriting the current clipboard content. The clipboard can later be reviewed in Boxer or manipulated as required.

4.7.21 User Lists

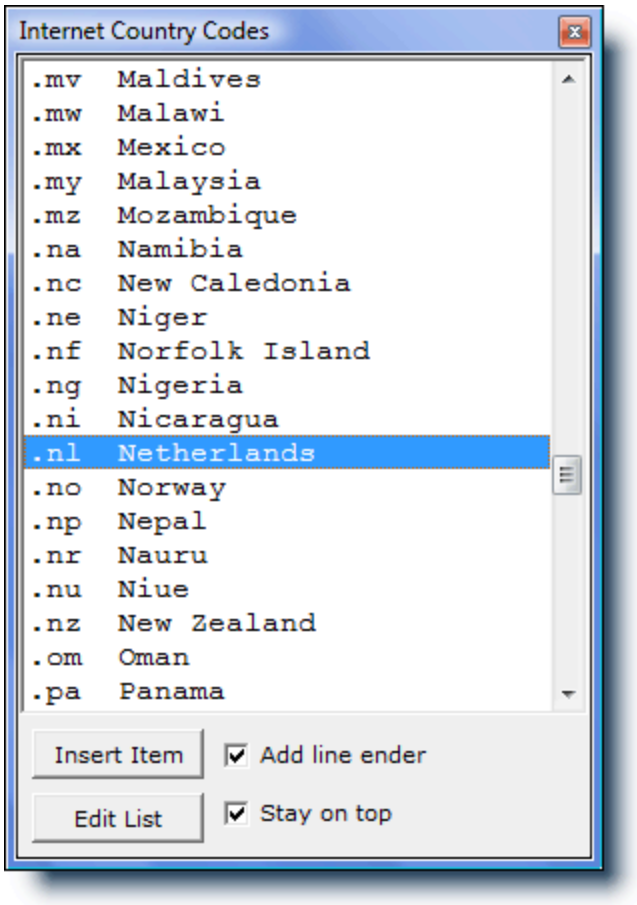
Menu: Tools > User Lists

Default Shortcut Key: none

Macro function: UserList()

The User Lists command provides access to a submenu of user-defined lists. Several example User Lists have been supplied with Boxer to suggest ways in which this feature might be used. You will no doubt think of many other ways.

Selecting a list from the User Lists submenu results in a popup window which displays the items in the list:



User Lists can be used for reference or to insert text strings into the file being edited. To insert a list item at the text cursor, double-click on the entry, highlight it and press *Enter*, or click the *Insert Item* button. If you would like a line ender to be added after inserting the item, use the *Add line ender* option.

Right-clicking on a selected item summons the User List context menu. The context menu provides options to insert the selected item, or to copy it to the current clipboard.

To advance quickly to an item in the list, enter its first letter from the keyboard.

To edit an existing list, click the *Edit List* button. To edit an empty list, simply select it from the User Lists submenu. In either case, the file which defines the list will be loaded into an editor window and can be edited in the usual way. The title of the list appears on the first line of the file and will be placed in the title bar of the popup window. The list items are placed one item per line, beginning on line two.

The maximum length of a User List item is 256 characters. The maximum length of a User List title is 40 characters.

The files which define User Lists are kept in Boxer's *data folder* in a subdirectory called 'User Lists' and are named `userlist.001`, `userlist.002`, etc.

If you prefer that the User List window remain atop other windows, select the *Stay on top* option. The User List windows are *non-modal* windows, which allows them to remain on-screen after *focus* has been returned to another editing window. Multiple User List windows can be opened simultaneously.

Email and URL Addresses
If a User List entry contains either an email or internet address, Boxer will launch the default email client or internet browser when the entry is double-clicked. This makes it possible for a User List to be used to create a list of email contacts or favorite websites. Email addresses can be entered in any of the following formats.

```
Boxer Software <sales@boxersoftware.com>
<sales@boxersoftware.com>
sales@boxersoftware.com
```

Mailto extensions can be used within an email address. They are appended to the email address following a question mark (`?`). Here are some examples:

```
sales@boxersoftware.com?cc=joe@mycompany.com
sales@boxersoftware.com?bcc=bill@mycompany.com
sales@boxersoftware.com?subject=Order a Site License for Boxer
sales@boxersoftware.com?body=this text will appear in the
message body
```

Multiple mailto extensions can be combined with the ampersand (`&`) symbol:

```
sales@boxersoftware.com?subject=Order  Boxer&cc=joe@mycompany.com
```

Note: Mailto extensions are not supported by all email clients. Experiment with your email client to learn its capabilities.

Sin Since an address can be launched with a double-click, the Enter key retains the function of inserting the text of the entry into the current file.

To ensure that special characters are displayed in the User List window as they will appear when inserted into the editor, the User List uses the same font as is used in the editor itself.

If a User List is left on-screen when Boxer is closed, it will be automatically reopened if the edit session is later restored.

4.7.22 User Lists -> Bring User Lists to Top

Menu: Tools > User Lists > Bring User Lists to Top

Default Shortcut Key: none

Macro function: BringUserListsToTop()

Use this command to bring all open User List windows to the top of the desktop.

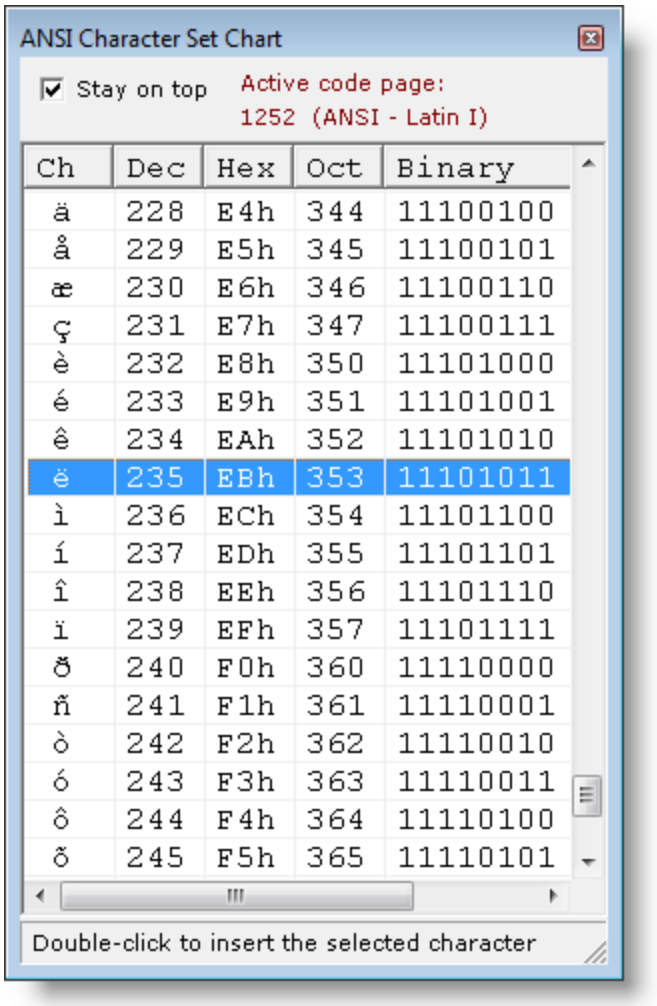
4.7.23 Reference Charts -> ANSI Chart

Menu: Tools > ANSI Chart

Default Shortcut Key: none

Macro function: ANSIChart()

The ANSI Chart command provides access to a popup chart which displays characters 0 to 255 in the ANSI character set. The character's visual representation is shown in the leftmost column, followed by the character value in *decimal*, *hexadecimal*, *octal* and *binary* formats. The active code page is also displayed at the top of the dialog:



Ch	Dec	Hex	Oct	Binary
ä	228	E4h	344	11100100
å	229	E5h	345	11100101
æ	230	E6h	346	11100110
ç	231	E7h	347	11100111
è	232	E8h	350	11101000
é	233	E9h	351	11101001
ê	234	EAh	352	11101010
ë	235	EBh	353	11101011
ì	236	ECh	354	11101100
í	237	EDh	355	11101101
î	238	EEh	356	11101110
ï	239	EFh	357	11101111
ð	240	F0h	360	11110000
ñ	241	F1h	361	11110001
ò	242	F2h	362	11110010
ó	243	F3h	363	11110011
ô	244	F4h	364	11110100
õ	245	F5h	365	11110101

To jump directly to a character of interest, simply press that character on the keyboard.

The ANSI Chart can be used to insert a character into the file being edited. Simply double click on the entry for the desired character or highlight the character in the chart and press *Enter*. When the need to insert a special character or symbol arises frequently, consider using the Insert Symbols feature rather than the ANSI Chart command. The Insert Symbols feature permits a defined character to be entered using a single keystroke.

Right-clicking on a selected item summons the ANSI Chart *context menu*. The context menu provides an option to copy the selected character to the current clipboard.

The ANSI Chart can also be used to convert between bases for values in the range 0 to 255. Simply locate the value to be converted in its proper column and read the converted value from the column of the new base.

If you prefer that the ANSI Chart remain atop other windows, select the *Stay on top* option. The ANSI Chart is a *non-modal* window, which allows it to remain on-screen after *focus* has been returned to another editing window.

 The ANSI Chart uses the same font and *code page* as the current screen font.

 If the ANSI Chart left on-screen when Boxer is closed, it will be automatically reopened if the edit session is later restored.

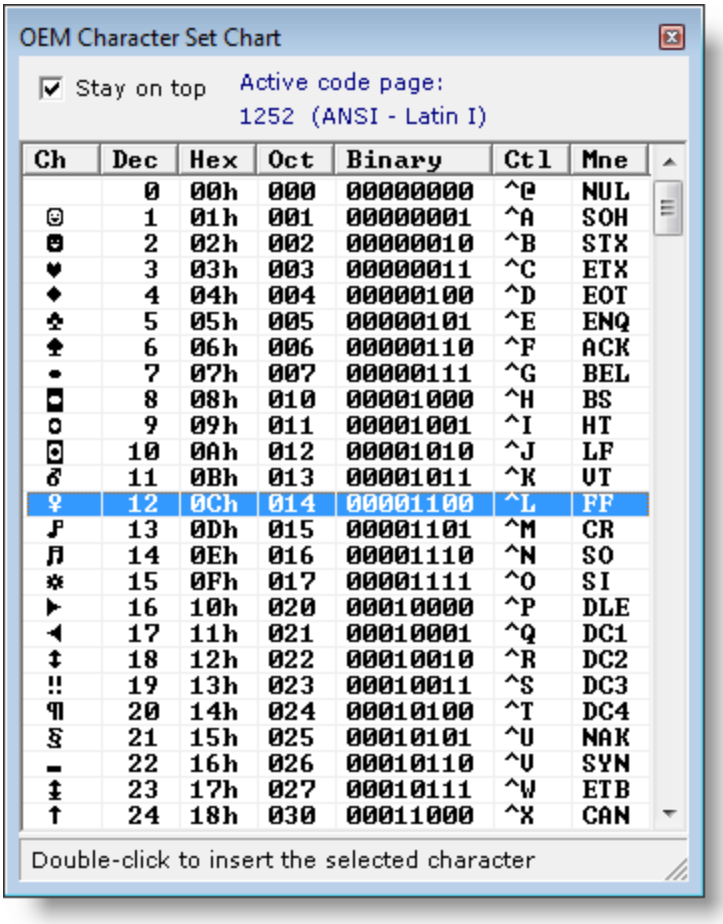
4.7.24 Reference Charts -> OEM Chart

Menu: Tools > OEM Chart

Default Shortcut Key: none

Macro function: OEMChart()

The OEM Chart command provides access to a popup chart which displays characters 0 to 255 in the OEM (ASCII) character set. The character's visual representation is shown in the leftmost column, followed by the character value in *decimal*, *hexadecimal*, *octal* and *binary* formats. In the two rightmost columns, the equivalent control letter sequence and mnemonic expression are shown for values in the range 0 to 31. The active code page is also displayed at the top of the dialog:



Ch	Dec	Hex	Oct	Binary	Ctl	Mne
	0	00h	000	00000000	^@	NUL
☺	1	01h	001	00000001	^A	SOH
☻	2	02h	002	00000010	^B	STX
♥	3	03h	003	00000011	^C	ETX
♦	4	04h	004	00000100	^D	EOT
♣	5	05h	005	00000101	^E	ENQ
♠	6	06h	006	00000110	^F	ACK
•	7	07h	007	00000111	^G	BEL
◘	8	08h	010	00001000	^H	BS
○	9	09h	011	00001001	^I	HT
◙	10	0Ah	012	00001010	^J	LF
♂	11	0Bh	013	00001011	^K	VT
♀	12	0Ch	014	00001100	^L	FF
♪	13	0Dh	015	00001101	^M	CR
♫	14	0Eh	016	00001110	^N	SO
☼	15	0Fh	017	00001111	^O	SI
►	16	10h	020	00010000	^P	DLE
◄	17	11h	021	00010001	^Q	DC1
↕	18	12h	022	00010010	^R	DC2
‼	19	13h	023	00010011	^S	DC3
¶	20	14h	024	00010100	^T	DC4
§	21	15h	025	00010101	^U	NAK
▬	22	16h	026	00010110	^V	SYN
↨	23	17h	027	00010111	^W	ETB
↑	24	18h	030	00011000	^X	CAN

To jump directly to a character of interest simply press that character on the keyboard.

The OEM Chart can be used to insert a character into the file being edited. Simply double click on the entry for the desired character, or highlight the character in the chart and press *Enter*. When the need to insert a special character or symbol arises frequently, consider using the Insert Symbols feature rather than the OEM Chart command. The Insert Symbols feature permits a defined character to be entered using a single keystroke.

Right-clicking on a selected item summons the OEM Chart context menu. The context menu provides an option to copy the selected character to the current clipboard.

The OEM Chart can also be used to convert between bases for values in the range 0 to 255. Simply locate the value to be converted in its proper column and read the converted value from the column of the new base.

If you prefer that the OEM Chart remain atop other windows, select the *Stay on top* option. The OEM Chart is a *non-modal* window, which allows it to remain on-screen after *focus* has been returned to another editing window.

If the OEM Chart is left on-screen when Boxer is closed, it will be automatically reopened if the edit session is later restored.

4.7.25 Reference Charts -> Value at Cursor

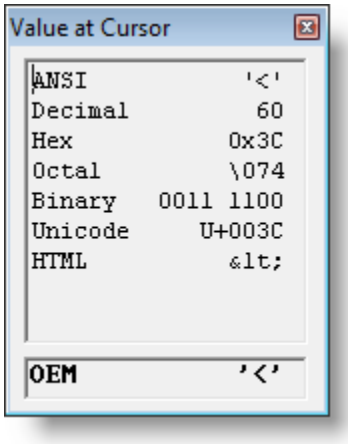
Menu: Tools > Value at Cursor

Default Shortcut Key: none

Macro function: ValueAtCursor()

The Value at Cursor command can be used to display information about the character at the text cursor. A popup window displays the character's ANSI representation and its character value in *decimal*, *hexadecimal*, *octal* and *binary* formats. The character's *Unicode* code point is also displayed. If the character has a named HTML entity, that name is displayed. Otherwise, the numeric HTML symbol is displayed.

A report for the less than symbol ('<') looks like this:



The dialog box will remain open until dismissed, and will continue to report on the character at the text cursor each time it changes.

This command can be especially useful for determining the value of characters which do not have a unique representation in the character set of the current Screen Font. For example, many ANSI fonts use an open box to represent all characters below the value 32 (Space), making it impossible to determine a character's value simply by looking at it.

The ANSI Chart and OEM Chart commands can be used to see a full listing of character values for each of these character encoding schemes.

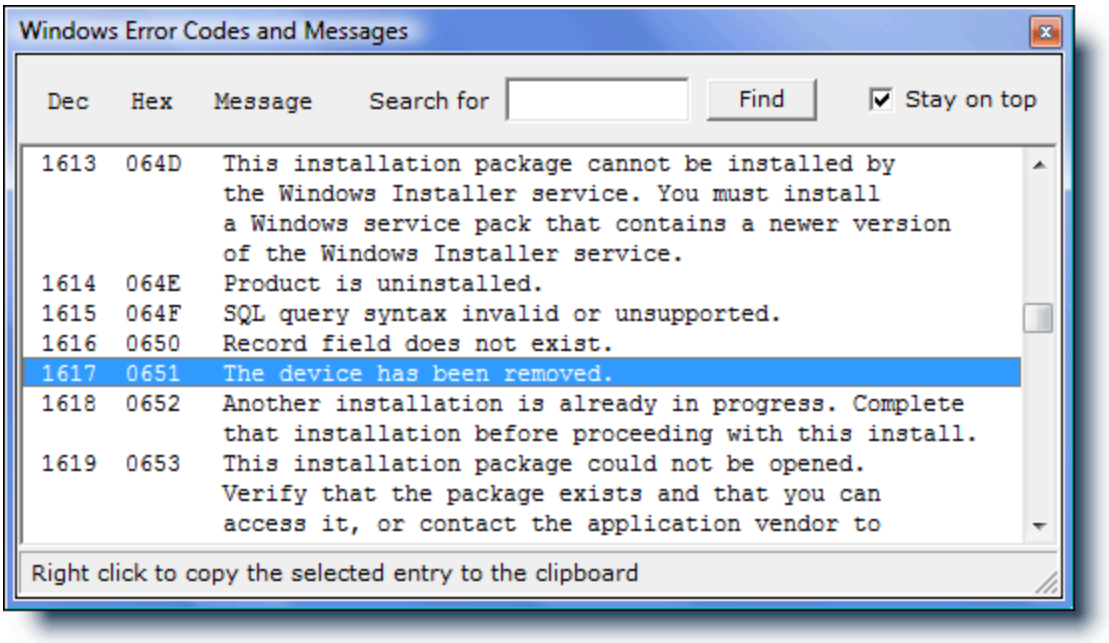
4.7.26 Reference Charts -> Error Chart

Menu: Tools > Error Chart

Default Shortcut Key: none

Macro function: ErrorChart()

The Error Chart command displays a popup list of Windows error codes and their associated messages. When errors are reported by the operating system--or by an application program--they will often reference a numeric error code. These reports frequently have insufficient information about the error which occurred. Boxer's Error Chart can be used by programmers--or by any users--to decode the meaning of Windows error codes.



The Error Chart can be searched by value or by any text which appears within the listing. Type the search string into the edit box provided and click *Find*. The *Find* button is also used to find the next occurrence of a string which has just been found.

Right-clicking on a selected item summons the Error Chart context menu. The context menu provides an option to copy the selected message to the current clipboard.

If you prefer that the Error Chart remain atop other windows, select the *Stay on top* option. The Error Chart is a *non-modal* window, which allows it to remain on-screen after *focus* has been returned to another editing window.

If the Error Chart is left on-screen when Boxer is closed, it will be automatically reopened if the edit session is later restored.

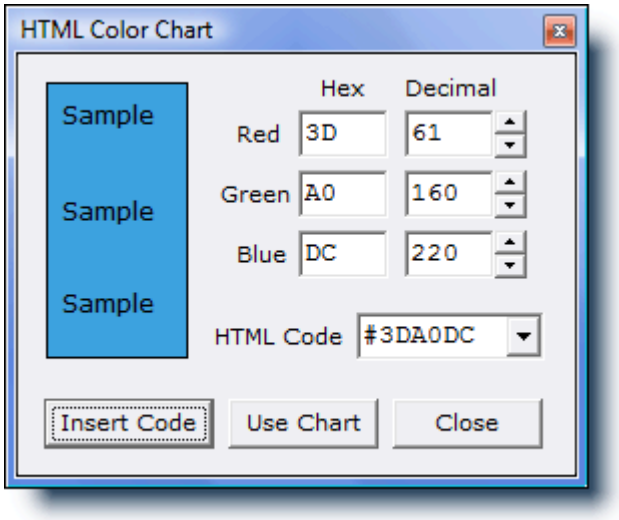
4.7.27 Reference Charts -> HTML Color Chart

Menu: Tools > HTML Color Chart

Default Shortcut Key: none

Macro function: ColorChart()

The HTML Color Chart command presents a pop-up dialog that allows color values to be viewed and adjusted by varying the Red, Green and Blue components. RGB values are shown in both *hexadecimal* and *decimal* format..



The current color is displayed in a rectangle at the left, along with sample text in black and white to show the contrast that would result for those color combinations. The value required to display the current color is shown in the box labeled *HTML Code*. The drop-down list holds a history list of recently used color codes. You can type an HTML color value directly into the *HTML Code* combobox, if you wish.

The *Use Chart* button can be used to summon the standard Windows color dialog so that a selection can be made from a color palette. The *Insert Code* button is used to insert the HTML Code into the current text file.

If the HTML Color Chart is left on screen when Boxer is closed, it will be automatically reopened if the edit session is later restored.

4.7.28 Templates

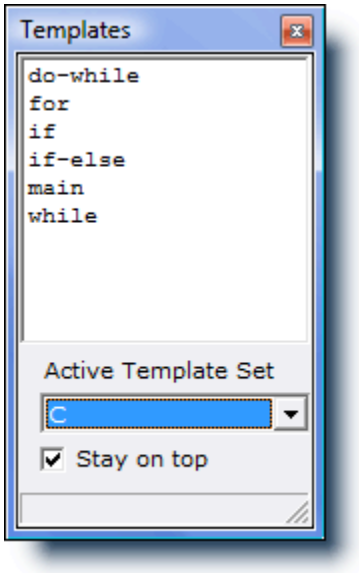
Menu: Tools > Templates

Default Shortcut Key: Ctrl+T

Macro function: Templates()

The Templates command is used to select a Template for insertion at the text cursor. Boxer's Templates are an excellent way to save and recall small pieces of text such as address blocks, copyright notices, programming language constructs, email addresses and any other text block which is used frequently during editing.

When the Insert Template command is issued, the Templates menu appears in a popup window:



The active Template Set is displayed at the bottom of the window, and the Templates within that set are displayed in the main listing area. Press the first letter of the Template name or cursor with the arrow keys to select the desired Template. Press *Enter* (or double click with the mouse) to insert the selected Template into the file.

Right-clicking on a selected item summons the Template context menu. The context menu provides options to insert the selected item, or to copy it to the current clipboard.

Template Sets and Templates are defined using the Configure | Templates command.

Text Cursor Placement
The Vertical Rule character (|) can be placed within a Template to dictate where the text cursor should be placed within the Template following its insertion. This allows, for example, programming code blocks to be defined in which the text cursor is placed between a pair of parentheses, ready for additional code to be typed.

Operating on Selected Text
The caret or circumflex character (^) can be placed within a Template to indicate that the template should operate on a text selection. A pair of examples will help to illustrate the power of this feature:

Example 1:
The template `<B>^</B>|` would cause the current text selection to be surrounded with HTML bold tags. The text cursor would be placed at the right of the closing bold tag.

Example 2:
The following Template:

```
<html>
<head>
|
</head>
```

```
<body>
^
</body>
</html>
```

could be run after using the Select All Text command to select the entire file. The effect would be to add the required HTML tags that help make an ordinary text file ready for viewing on the Internet. The text cursor would be place between the <head> and </head> tags, awaiting a title for the document.

Unindenting within a Template
If you need to unindent within a defined Template, use the tilde character (~) to designate the point at which the Backspace command should occur. The tilde will not be recognized in this way unless the *Insert as if typed from the keyboard* option is in force (see Configure | Templates).

If you need to insert one of the special characters (~, | or ^) into a template in its textual form, use either ~~, || or ^^.

To ensure that special characters are displayed in the Template window as they will appear when inserted into the editor, the Template window uses the same font as is used in the editor itself.

If the need arises to insert a single character which is not easily typed from the keyboard, consider using the Insert Symbols feature rather than defining a single character Template. The Insert Symbols feature permits a defined character to be entered using a single keystroke.

If the Template window is left on-screen when Boxer is closed, it will be automatically reopened if the edit session is later restored.

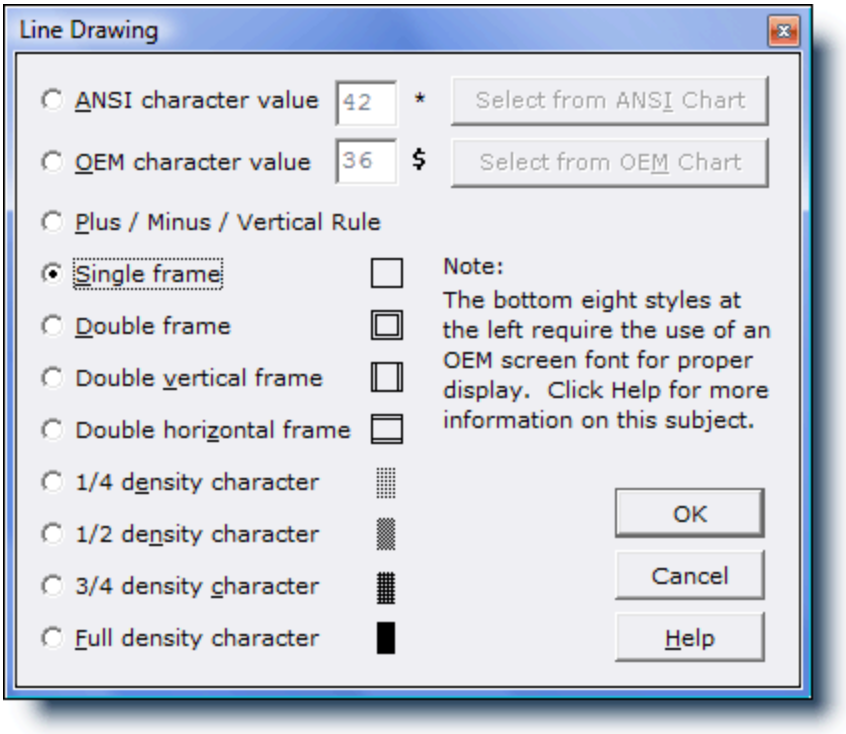
4.7.29 Line Drawing

Menu: Tools > Line Drawing

Default Shortcut Key: Ctrl+F12

Macro function: LineDrawing()

The Line Drawing command can be used to enter a mode in which the arrow keys are used to draw lines in a selected frame style, or with a specified character. A dialog box appears from which the drawing style is selected:



A drawing style is selected by checking the radio button which corresponds to the desired style. Options are also provided to draw with some other character, and either the ANSI Chart or OEM Chartcan be summoned to assist in character selection.

When using an OEM Screen Font, several additional drawing styles may be used. The bottom eight styles offered in the dialog box require the use of an OEM screen font for proper display. The ANSI character set does not contain these characters, so using these styles with an ANSI Screen Font will produce undesirable results.

Once OK is clicked, Line Drawing mode is active. Use the arrow keys to create lines or boxes as desired. Use *Esc* to cancel Line Drawing mode.

When drawing atop lines which contain Tab characters, the Tabs will be automatically converted to Spaces to ensure proper display. When Line Drawing occurs past the end of a line, the line will be automatically extended with Spaces. If the end of file is encountered, additional blank lines will be added automatically so that Line Drawing can proceed.

When printing files which contain drawing characters from the OEM character set, be sure to use a Printer Font which also uses the OEM character set.

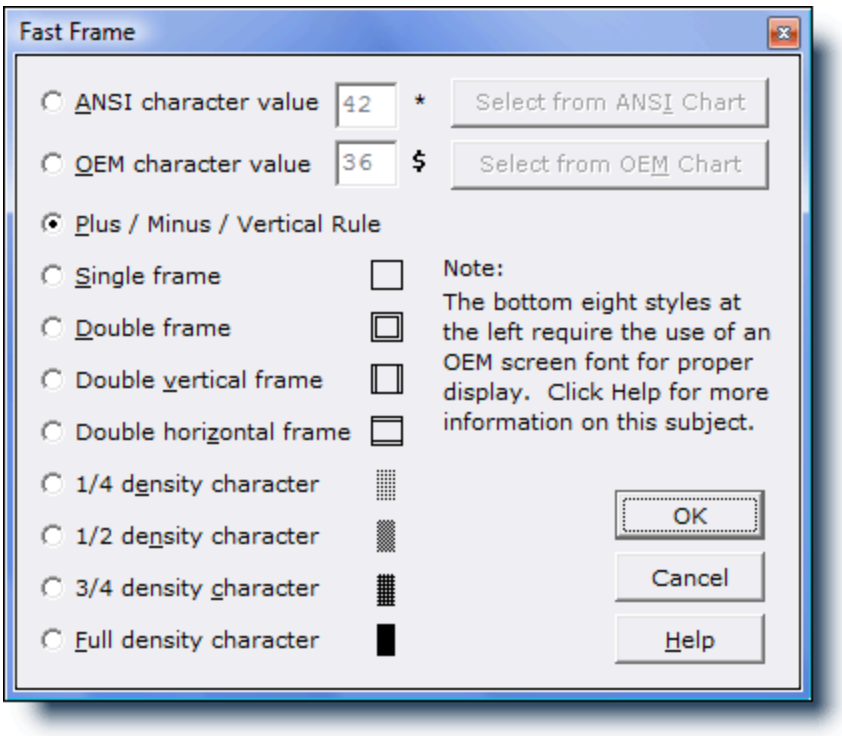
4.7.30 Fast Frame

Menu: Tools > Fast Frame

Default Shortcut Key: Alt+F12

Macro function: FastFrame()

The Fast Frame command can be used to frame a columnar-selected rectangle with a chosen frame style, or with a specified character. A dialog box appears from which the frame style is selected:



A frame style is selected by checking the radio button which corresponds to the desired style. Options are also provided to frame the selected area with some other character, and either the ANSI Chart or OEM Chart can be summoned to assist in character selection.

When using an OEM Screen Font, several additional frame styles may be used. The bottom eight styles offered in the dialog box require the use of an OEM screen font for proper display. The ANSI character set does not contain these characters, so using these styles with an ANSI Screen Font will produce undesirable results.

Once OK is clicked, the selected area is automatically framed with the chosen frame style. The frame is applied to the outside of the selected area. If the left edge of the selection lies in column one, the lines in the selected range will be pushed right by one column to make room for the frame. If Tab characters appear within the selection, they will be automatically converted to Spaces to ensure proper display after framing.

☞ When printing files which contain drawing characters from the OEM character set, be sure to use a Printer Font which also uses the OEM character set.

4.8 Project Menu

4.8.1 New

Menu: Project > New

Default Shortcut Key: none

Macro function: ProjectNew()

Boxer's Project feature provides a means of simultaneously loading a collection of files, and restoring those files to their previous editing states. The Project | New command creates a project file containing the names and editing options of all files that are currently open.

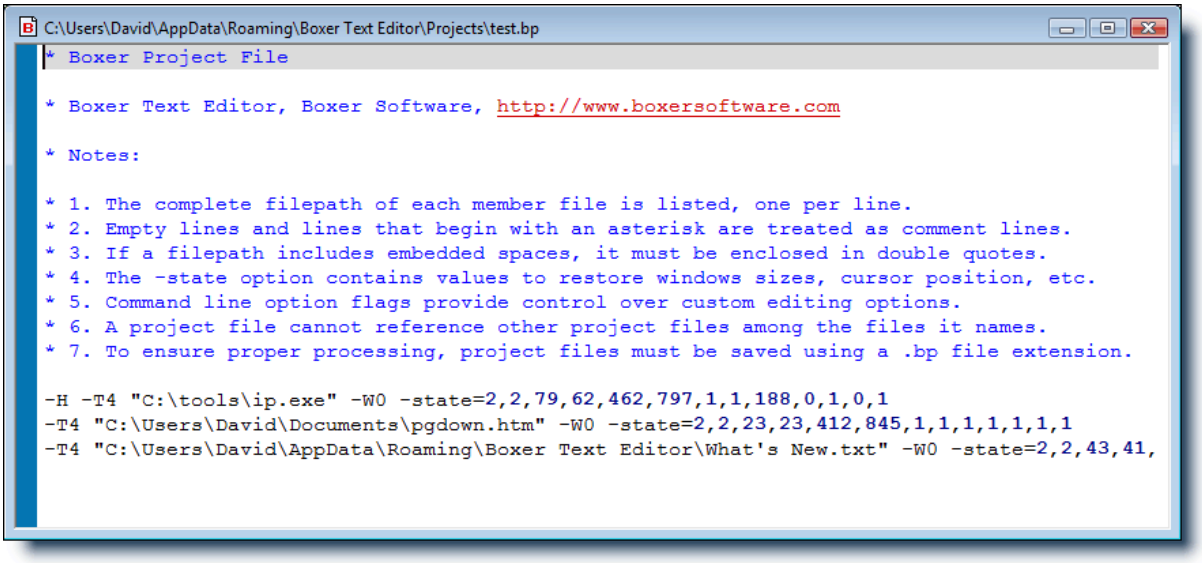
The following editing options are maintained within a project file:

- window sizes and positions
- cursor location
- active file
- bookmarks
- tab stops
- typing wrap mode
- hex editing mode
- file tab arrangement

Once a project is open, the Add One, Add All and Remove commands can be used to add and remove files from the active project. The Update One and Update All commands can be used keep the project file up-to-date with the current editing options. To keep the editing options up-to-date automatically, use the Auto-Update command.

When a project file is named on Boxer's command line, or when the icon of a project file is dragged and dropped onto the Boxer window (or its icon), all of the files named within that file will be loaded for editing. If you need to edit the content of the project file itself, use the Edit Active or Edit Other command, as may be appropriate.

The project file is a text file that contains an informative series of comments that explains the use of project files:



In its simplest form, a project file is simply a text file whose file extension is .BP and whose content consists of a list of filenames, one per line. Empty lines can be used freely within a project file to separate filenames as may be appropriate. Lines beginning with an asterisk (*) will be considered comment lines, and will not be processed.

A project file can be used to maintain a list of filenames that relates to a given project or document set, and to open those files quickly. For best results, the full pathname--including the drive designator and directory path--should be used. This will ensure that the project file functions properly regardless of the default directory in force at the time it is used.

FTP filepaths can be placed within project files. See the FTP Open command for more information.

Project files cannot be nested; if a project file is named within another project file an error will occur.

If a file is open for read-only editing, that file's entry in the project file will be automatically created with the -R command line option flag that designates read-only status. Likewise, if a file is open for hex mode viewing, its entry will be created with the -H command line option flag.

Within a project file, filepaths can be preceded with "exec:" to indicate that they be opened using their default application. This allows other files that are associated with a project to be opened when the project opens in Boxer. For example, a project file's entries might be:

```
c:\myproject\source\main.cpp
exec:c:\myproject\docs\updates.doc
exec:http://www.mysite.com/index.htm
exec:c:\myproject\bitmaps\project_logo.bmp
```

The first file would open in Boxer, while the next three files would be opened by the applications that are associated with their respective file types: DOC/HTM/BMP. If the filepath to be opened contains embedded spaces, the entire line must be surrounded in double quotes:

"exec:c:\my project\docs\monthly updates.doc"

A project file can be designated on Boxer's command line using the `-P` command line option flag.

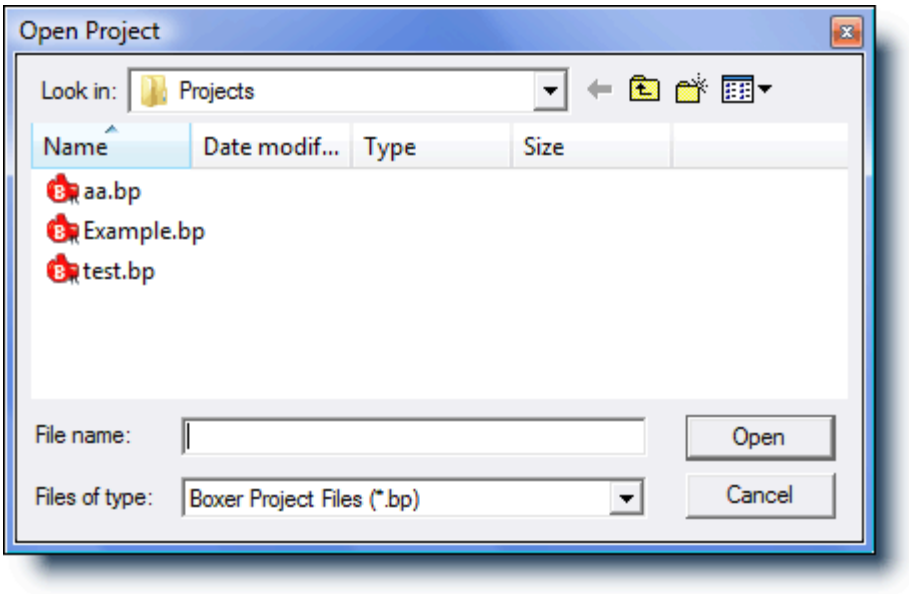
4.8.2 Open

Menu: Project > Open

Default Shortcut Key: none

Macro function: ProjectOpen()

The Open Project Command is used to open an existing project file. When a project file is opened, all of the files named therein are open for editing. If you need to edit the content of the project file itself, use the Edit Active or Edit Other command, as may be appropriate.



See the Project | New command for full details about Boxer's project file feature.

4.8.3 Close

Menu: Project > Close

Default Shortcut Key: none

Macro function: ProjectClose()

Use the Project | Close command to close the current project. All files associated with the active project will be closed.

 See the Project | New command for full details about Boxer's project file feature.

4.8.4 Delete

Menu: Project > Delete

Default Shortcut Key: none

Macro function: ProjectDelete()

Use the Project | Delete command to delete a selected project file. This command deletes a project file. It does not delete the files named within that file.

 See the Project | New command for full details about Boxer's project file feature.

4.8.5 Add One

Menu: Project > Add One

Default Shortcut Key: none

Macro function: ProjectAddOne()

Use the Add One command to add the current file to the active project.

 See the Project | New command for full details about Boxer's project file feature.

4.8.6 Add All

Menu: Project > Add All

Default Shortcut Key: none

Macro function: ProjectAddAll()

Use the Add All command to add all open files to the active project.

 This command can be used safely even when some of the open files are known to already reside within the active project. Duplicate entries will not result.

 See the Project | New command for full details about Boxer's project file feature.

4.8.7 Remove

Menu: Project > Remove

Default Shortcut Key: none

Macro function: ProjectRemove()

Use the Remove command to remove the current file from the active project.

 See the Project | New command for full details about Boxer's project file feature.

4.8.8 Update One

Menu: Project > Update One

Default Shortcut Key: none

Macro function: ProjectUpdateOne()

Use the Update One command to update the active project file with the current editing options for the active file.

The project file stores the following information about the files contained in the project:

- window sizes and positions
- cursor location
- active file
- bookmarks
- tab stops
- typing wrap mode
- hex editing mode
- file tab arrangement

 If you would like the project file to be updated automatically for all member files, use the Project | Auto-Update feature.

4.8.9 Update All

Menu: Project > Update All

Default Shortcut Key: none

Macro function: ProjectUpdateAll()

Use the Update All command to update the active project file with the current editing options for *all* files within the project.

The project file stores the following information about the files contained in the project:

- window sizes and positions
- cursor location
- active file
- bookmarks
- tab stops
- typing wrap mode
- hex editing mode
- file tab arrangement

 If you would like the project file to be updated automatically for all member files, use the Project | Auto-Update feature.

4.8.10 Auto-Update

Menu: Project > Auto-Update

Default Shortcut Key: none

Macro function: ProjectAutoUpdatel()

This option can be used to keep the editing options of the active project updated automatically. As windows sizes, window locations, bookmarks, cursor locations and other file-specific options change, the project file will be updated automatically. Each project is permitted to have a different Auto-Update setting, if desired.

If you prefer to maintain project settings manually, the Update One and/or Update All commands can be used to manually update the editing options for the current file, or all open files, on an as-needed basis.

4.8.11 Edit Active

Menu: Project > Edit Active

Default Shortcut Key: none

Macro function: ProjectEditActive()

Use the Edit Active command to open the active project file for editing. This command opens a project file as a text file; it does not open the files named within that project file. To open the files within a project file, use the Project | Open command.

 See the Project | New command for full details about Boxer's project file feature.

4.8.12 Edit Other

Menu: Project > Edit Other

Default Shortcut Key: none

Macro function: ProjectEditOther()

Use the Edit Other command to open a selected project file for editing. This command opens a project file as a text file; it does not open the files named within that project file. To open the files within a project file, use the Project | Open command.

☞ See the Project | New command for full details about Boxer's project file feature.

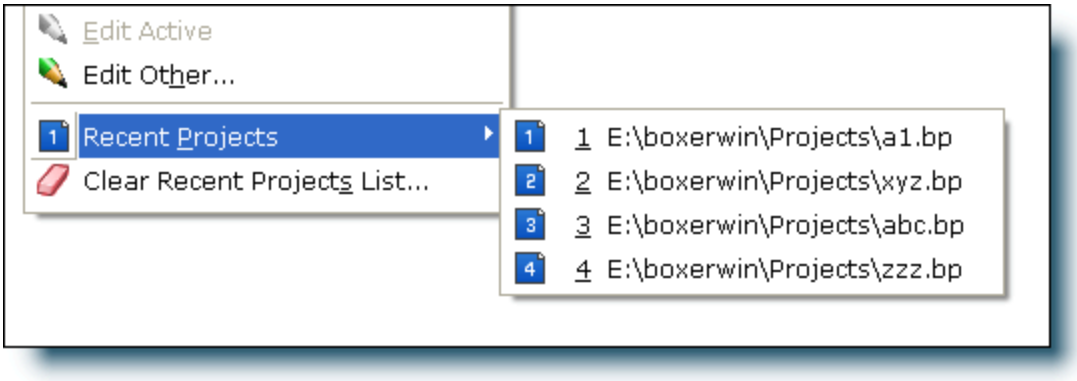
4.8.13 Recent Projects

Menu: Project

Default Shortcut Key: not applicable

Macro function: OpenRecentProject()

The Recent Projects list appears near the bottom of the Projects menu. Each time a projects is opened, its name is added to the list. If necessary, the eldest entry is bumped from the list. This list makes it easy to recall projects which were recently opened without the need to use the Project | Open command. The projects are displayed with a 'hot' number to their left, so that *Alt+P*, *P*, followed by the number, will load the named project.



Up to 16 recent projects can appear on the Recent Projects list.

Long project names will be shortened for display if the relevant option on the Configure | Preferences | Display options page is checked.

4.8.14 Clear Recent Projects List

Menu: Project > Clear Recent Projects List

Default Shortcut Key: none

Macro function: ClearRecentProjectsList()

Use this command to clear the record of recently accessed projects from the Recent Project submenu.

☞ See the Project | New command for full details about Boxer's project file feature.

4.9 Configure Menu

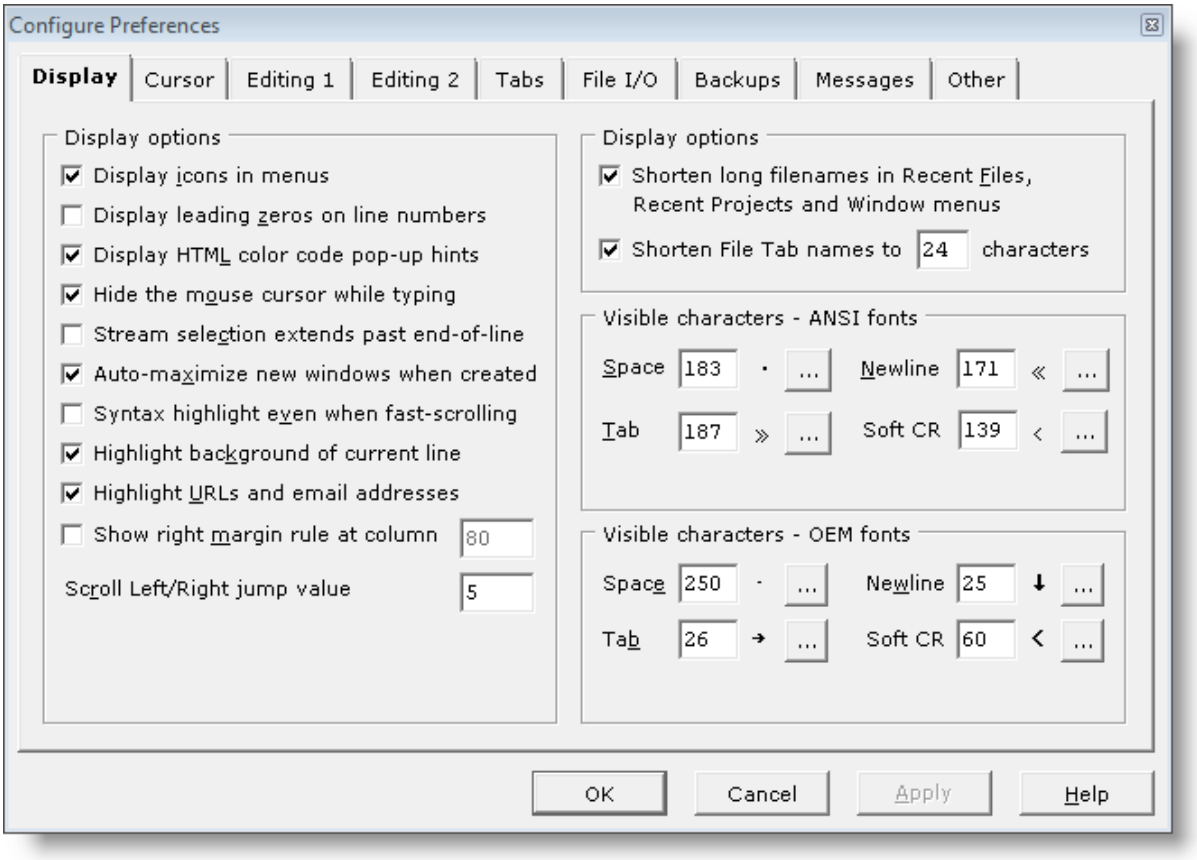
4.9.1 Preferences - Display

Menu: Configure > Preferences

Default Shortcut Key: none

Macro function: ConfigurePreferences()

The Display page of the Configure Preferences dialog box contains options related to the appearance of Boxer's screen and the display of edited files:



Display Options

Display icons in menus
Use this option to control the display of icons within the main menus.

Display leading zeros on line numbers
This option controls whether or not leading zeros will be used when the View | Line Numbers command is in use.

Display HTML Color Code pop-up hints
This option controls whether or not Boxer will display a pop-up HTML Color Code Hint when the mouse cursor hovers atop an HTML Color Code (such as `color="#FF3B80"` or `color="DarkSlateBlue"`). The pop-up box shows the color associated with the color sequence below the mouse cursor.

Hide the mouse cursor while typing
Use this option to control whether or not the mouse cursor will be hidden while text is being entered from the keyboard. If hidden, the mouse cursor will be redisplayed when it is moved.

Stream selection extends past end-of-line
This option controls the way in which a multi-line Stream selection is displayed on-screen. When this option is active, selected lines will be highlighted all the way to the right edge of the window, even when lines are shorter than the window's width. When this option is inactive, selected lines will be highlighted only up to the end of each line. This option has no effect on the text included within a selection. It is simply a display option.

Auto-maximize new windows when created
Use this option to cause new windows created with File | New or File | Open to be opened in maximized form.

Syntax highlight even when fast-scrolling
This option controls whether or not Boxer will perform Syntax Highlighting while a file is scrolling rapidly, due to a keyboard key being held down. Scrolling will be faster if Boxer is allowed to suspend Syntax Highlighting in this situation. The screen will be updated instantly when the scrolling key is released.

Highlight background of current line
Use this option to control whether or not the background of the current will be displayed in a different color. Doing so can make it easier to locate the current line. The Configure Colors command can be used to select the background color used.

Highlight URLs and email addresses
This option controls whether or not Boxer will apply coloration to URLs and email addresses when they are encountered within ordinary text files. The color and font style used to highlight an address can be controlled with the Configure | Colors command. Disabling highlighting also disables the ability to double-click on these addresses in order to launch an internet browser or email client. In such case, the Open File in Browser and Open Email at Cursor commands could be used instead.

Show right margin rule at column...
Use this option to control the display of the Right Margin Rule, and to set the column at

which the rule is displayed.

Scroll Left/Right jump value
This option controls the number of columns that the Scroll Left and Scroll Right commands will jump by when panning the screen left of right..

Shorten long filenames in File/Project/Window menus
Use this option to control whether or not long filenames will be shortened when they appear in the either the Recent Files, Recent Projects or Window menus. If this option is active, long file names will be shortened whenever they exceed 60 characters in length.

Shorten File Tab names to *n* characters
Use this option to control whether or not the filenames displayed in Boxer's File Tabs will be shortened when they exceed a specified length. When filename shortening is required, as many as four characters will be retained from the file extension, with the balance of characters being retained from the left side of the filename. Missing characters in the middle of the filename will be replaced by three dots (. . .).

Visible characters - ANSI fonts

Space value
Tab value
Newline value
These options can be used to designate the characters which will be used for Visible Spaces display when an ANSI Screen Font is in use. Use the button with the ellipsis (...) to select a character from the ANSI Chart.

Visible characters - OEM fonts

Space value
Tab value
Newline value
These options can be used to designate the characters which will be used for Visible Spaces display when an OEM Screen Font is in use. Use the button with the ellipsis (...) to select a character from the OEM Chart.

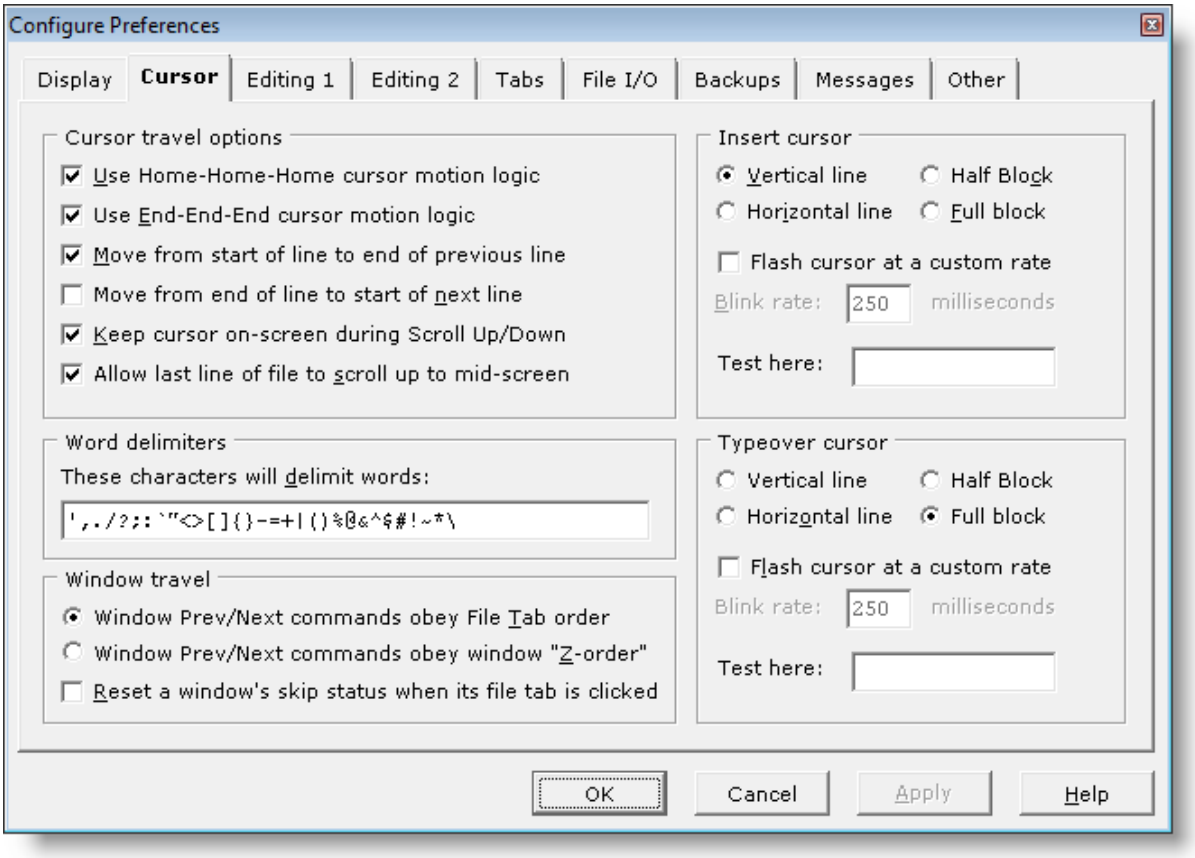
4.9.2 Preferences - Cursor

Menu: Configure > Preferences

Default Shortcut Key: none

Macro function: ConfigurePreferences()

The Cursor page of the Configure Preferences dialog box contains options which can be used to control cursor movement within the file:



Cursor Travel Options

Use Home-Home-Home cursor motion logic
When this option is active the function of the *Home* key becomes dependent on the number of times *Home* has been pressed. When *Home* is first pressed, the text cursor is moved to the beginning of the current line. When *Home* is pressed again, the cursor is moved to the first line in the window. When *Home* is pressed a third time, the cursor is moved to the first line in the file.

☞ If the *Home* key is not assigned to *Start of Line*, *Home-Home-Home* functionality becomes unavailable. Its function is tied directly to the *Home* key, and not to other keys which might be assigned to *Start of Line*.

Use End-End-End cursor motion logic
When this option is active the function of the *End* key becomes dependent on the number of times *End* has been pressed. When *End* is first pressed, the text cursor is moved to the end of the current line. When *End* is pressed again, the cursor is moved to the last line in the window. When *End* is pressed a third time, the cursor is moved to the last line in the file.

☞ If the *End* key is not assigned to *End of Line*, *End-End-End* functionality becomes unavailable. Its function is tied directly to the *End* key, and not to other keys which might be assigned to *End of Line*.

Move from start of line to end of previous line
Use this option to permit the text cursor to move from the start of the current line to the end of the previous line when the *Left Arrow* is pressed. When this option is inactive, pressing the *Left Arrow* in column one will result in no cursor movement.

Move from end of line to start of next line
Use this option to force the text cursor to move from the end of the current line to the start of the next line when the *Right Arrow* is pressed. When this option is inactive, the cursor is allowed to travel rightward past the end of a line.

This option also affects the way Boxer behaves when the *Up Arrow* and *Down Arrow* are used to cursor across lines of varying lengths. If this option is active, the column of the text cursor will be adjusted when moving onto a line that is shorter than the current column. If this option is inactive, the text cursor column will be maintained when moving from line to line.

Keep cursor on-screen during Scroll Up/Down
Use this option to control how the position of the text cursor is treated when using the Scroll Up and/or Scroll Down commands. When this option is checked, the text cursor will be moved (if necessary) to keep it on-screen while scrolling. When this option is not checked the cursor position will be maintained even when the line containing the text cursor is scrolled outside the view of the window.

Allow last line of file to scroll up to mid-screen
When this option is on, using the Down arrow to scroll to end of file will cause the last line of the file to appear as high as mid-screen. When this option is off, the last line of the file will not scroll up past the bottom of the window.

Word delimiters

These characters will delimit words
This option can be used to designate the characters which are considered to be *word delimiters*. Word delimiters are those characters which serve to separate one word from another. It may be desirable to add or remove symbols from the default delimiter list in order to improve the behavior of the *Word Left* and *Word Right* commands within certain types of file.

The word delimiter list is used by various commands to determine the extent of their operation. Among these commands are Word Left, Word Right, Delete Previous Word, Delete Next Word, Swap Words, Open Filename at Cursor, Open URL at Cursor and Open Email at Cursor.

Window Travel

Window Previous/Next command obey File Tab order
Window Previous/Next command obey window 'Z-order'
Use these options to control how Boxer responds to the Window Previous and Window Next commands. When *File Tab order* is selected, the window commands will use the ordering of the File Tabs to determine which window to move to. When the *Z-order* option is selected, window movement will be determined according to an order

maintained by Windows. A window is promoted in the Z-order when it is made current. Less frequently used windows will gradually fall to the bottom of the z-order.

Reset a window's skip status when its file tab is clicked
When this option is on, clicking on a file tab will cause its skip status to be reset to normal.

Insert cursor and Typeover cursor

The shape and flash rate can be set independently for the Insert and Typeover cursors.

Vertical Line
Use this option to set the text cursor to a thin vertical line which sits at the left edge of the character cell.

Horizontal Line
Use this option to set the text cursor to a horizontal line which sits at the base of the character cell.

Half Block
Use this option to set the text cursor to a block which occupies the lower half of the character cell.

Full Block
Use this option to set the text cursor to a block which occupies the full character cell.

Flash cursor at a custom rate
Use this option to set the rate at which the cursor flashes. A *millisecond* is one thousandth of a second.

Test here
Use this edit box to test the new shape and flash rate.


Changes made to the text cursor apply only to Boxer, and do not affect other applications.

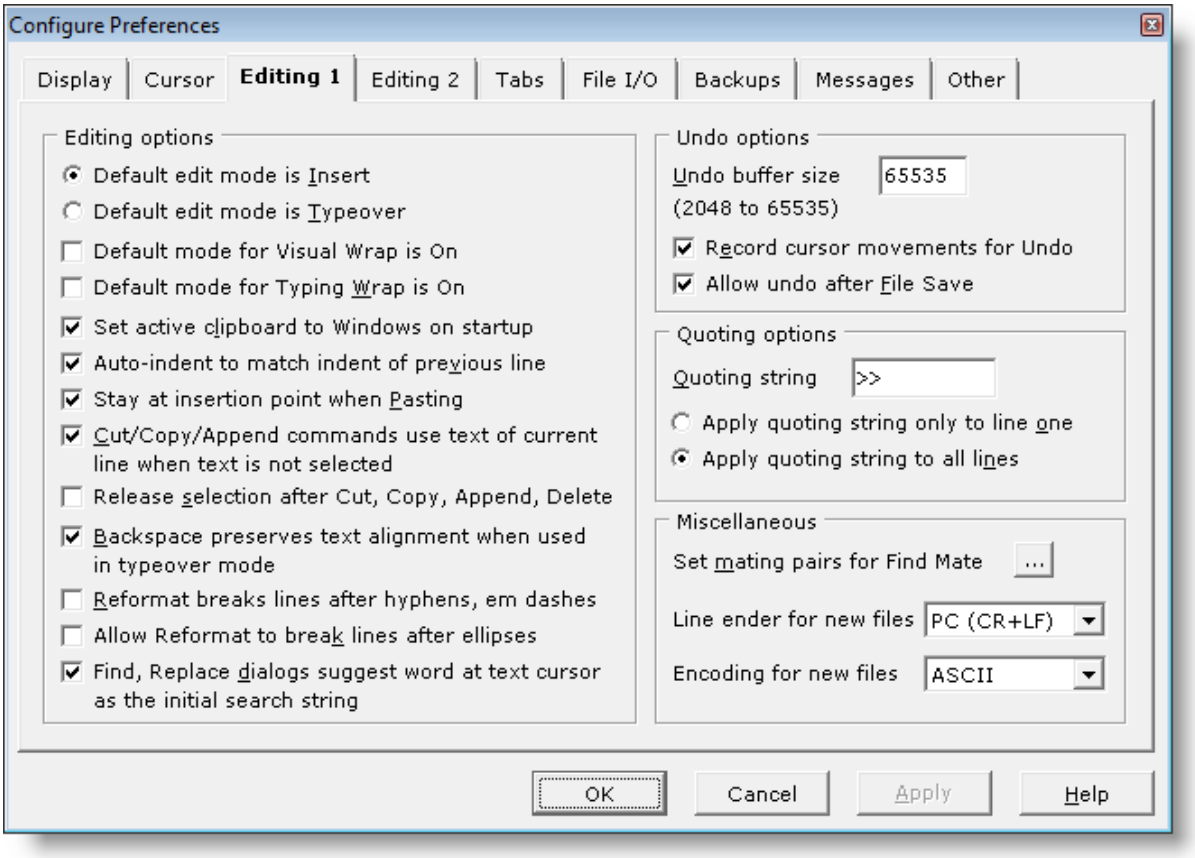
4.9.3 Preferences - Editing 1

Menu: Configure > Preferences

Default Shortcut Key: none

Macro function: ConfigurePreferences()

The Editing 1 page of the Configure Preferences dialog box contains options which relate to the editing of text files:



Editing options

Default edit mode is Insert
Use this option to set the default edit mode to *Insert*. In Insert mode, existing text is pushed rightward to make room for characters which are typed from the keyboard. Note that this option causes newly created windows to begin in Insert mode, but it does not change the edit mode of any editor windows that may already be open.

Default edit mode is Typeover
Use this option to set the default edit mode to *Typeover*. In Typeover mode, characters which are typed from the keyboard replace existing text. Note that this option causes newly created windows to begin in Typeover mode, but it does not change the edit mode of any editor windows that may already be open.

Default mode for Visual Wrap mode is On
Use this option to change the default Visual Wrap setting to on. Note that this option causes newly created windows to begin with Visual Wrap on, but it does not change the Visual Wrap mode of any editor windows that may already be open. The Visual Wrap command can always be used to change the setting for the current editor window, independent of this option.

Default mode for Typing Wrap mode is On
Use this option to change the default Typing Wrap setting to on. Note that this option causes newly created windows to begin with Typing Wrap on, but it does not change the

Typing Wrap mode of any editor windows that may already be open. The Typing Wrap command can always be used to change the setting for the current editor window, independent of this option.

Set active clipboard to Windows on startup
When checked, this options ensures that the active clipboard will be restored to the Windows clipboard each time Boxer is started.

Auto-indent to match indent of previous line
Use this option to enable *Auto-indent*. When Auto-Indent is active, pressing *Enter* at the end of a line will place the text cursor on a new line below with an indent level equal to that of the line above.

Stay at insertion point when Pasting
Use this option to cause the text cursor to remain at the point of insertion following a Paste operation. If inactive, the text cursor is placed at the end of the text which was pasted.

Cut/Copy/Append commands use text of current line when text is not selected
This option permits the Cut, Copy, Append and Cut Append commands to operate on the current line as though it were selected. Simply issue the desired command from any point on the line and the operation will be performed as though the whole line were selected.

Release selection after Cut, Copy, Append, Delete
When selected, this options causes a text selection to be automatically released after a clipboard operation is performed.

Backspace preserves text alignment when used in Typeover mode
Use this option to make the Backspace key overwrite with spaces when used in Typeover mode.

Allow Reformat to break lines after hyphens and em-dashes
When selected, this option allows the Paragraph | Reformat command to break a line after a hyphen (-) or an em-dash (--).

Allow Reformat to break lines after ellipses
When selected, this option allows the Paragraph | Reformat command to break a line after an ellipsis (...).

Find, Replace dialogs suggest word at text cursor as the initial search string
When selected, this option controls whether the Find, Replace, Find Text in Disk Files and Replace Line Enders dialogs will insert the word at the text cursor into the find edit box.

Undo Options

Undo buffer size
Use this command to set the buffer size used by the Undo command. Values between 2048 and 65535 are permitted. This value represents the amount of memory (in bytes) which is reserved for tracking undo information.

The default value is 65535, which is also the maximum value. There is little reason to select smaller values, as the memory cost is small compared to the utility that Undo provides.

Allow undo after File Save
Use this option to indicate that the Undo command should remain operable after the Save command is used, thereby allowing changes which occurred before Save to be undone. If this option is inactive, the Save command has the effect of the Clear Undo command, since Undo information is lost for changes made before the save.

Quoting Options

Quoting String
Use this option to specify the symbol (or symbols) which are to be used by the Quote and Reformat command during its operation.

☞ The Quote and Reformat command makes use of the Reformat command internally during its operation. As is noted in the Reformat command, lines beginning with a period (.) are treated as blank lines in order to recognize text markup tags. As a result, the use of a quoting string that begins with a period will not produce the desired results, and should be avoided. All other symbols and characters are permissible.

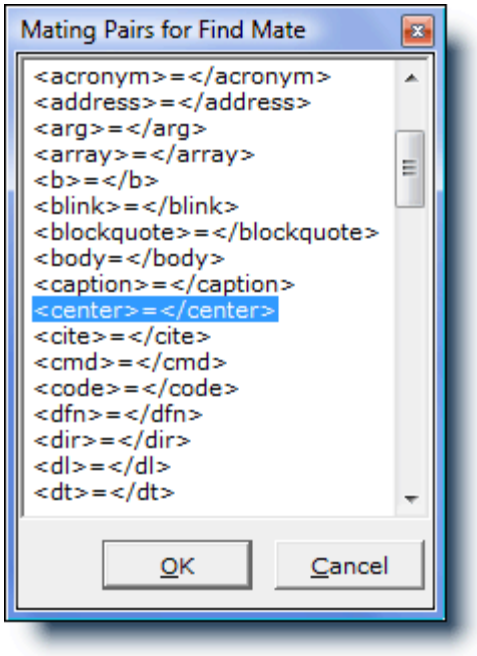
Apply quoting string only to line one
Use this option to designate that the Quote and Reformat command apply the *Quoting String* to the first line of the paragraph quoted.

Apply quoting string to all lines
Use this option to designate that the Quote and Reformat command apply the *Quoting String* to all lines within the paragraph quoted.

Miscellaneous

Set mating pairs for Find Mate
This option can be used to edit the mating pairs which are used by the Find Mate command. The Find Mate command is used to jump quickly from a parenthetical element at the text cursor to its mate.

When the ellipsis (...) button is clicked, a small edit window appears which contains the currently defined pairs:



Each mating pair resides on a single line, with the equal sign (=) being used to separate the opening string from the closing string. Pairs can be removed from the list, or new pairs can be added. Click *OK* to save the changes.

Line ender for new files
This option can be used to set the default line ender type for newly created files. Choose from PC, Macintosh or Unix style line enders. A file's line ender can also be changed from the File | Properties dialog.

Encoding for new files
This option can be used to set the default file encoding format for newly created files. Choose from ASCII, UTF-8, UTF-16 little endian or UTF-16 big endian. A file's encoding format can also be changed from the File | Properties dialog.

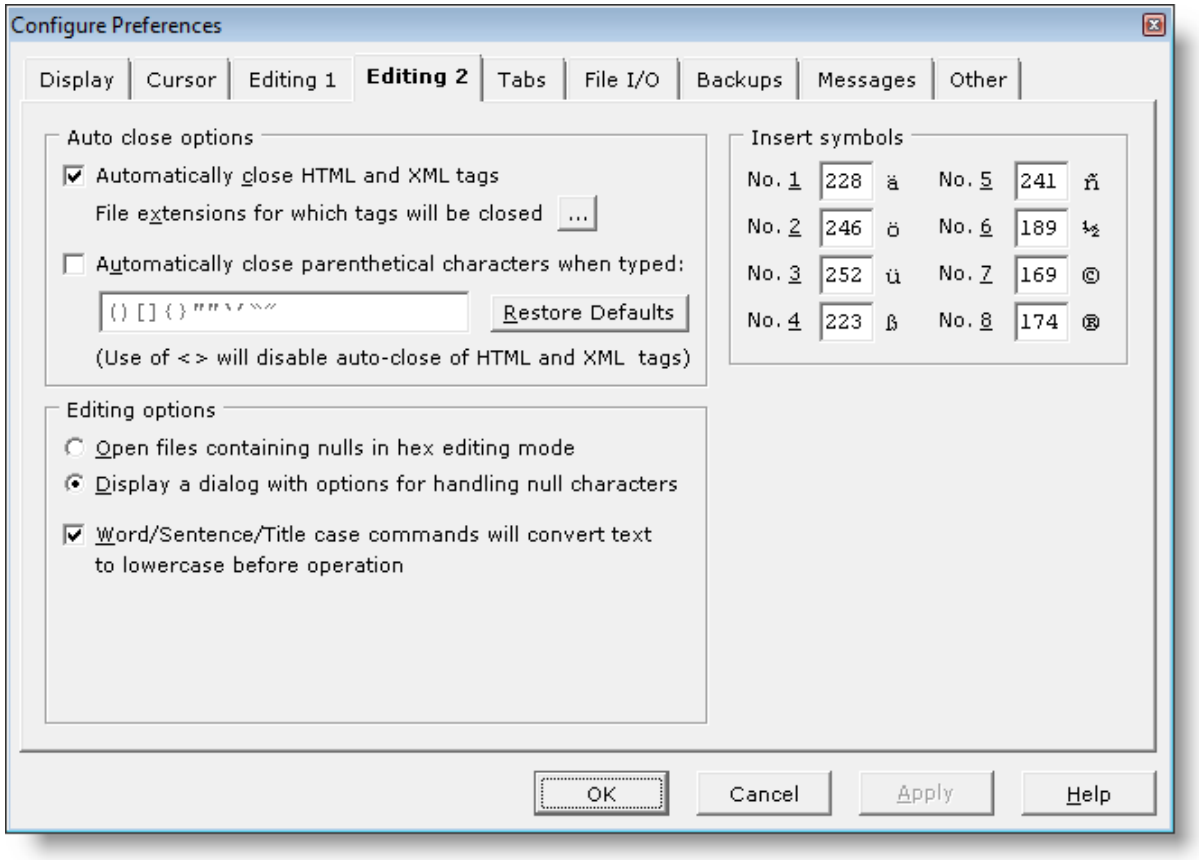
4.9.4 Preferences - Editing 2

Menu: Configure > Preferences

Default Shortcut Key: none

Macro function: ConfigurePreferences()

The Editing 2 page of the Configure Preferences dialog box contains options which relate to the editing of text files:



Auto close options

Auto Close HTML, XML tags
This option can be used to enable or disable Boxer's Auto Tag Close feature. When enabled, and when an eligible file type is being edited, Boxer will automatically create the closing tag and place the cursor between the tags. For example, when you type `<center>`, Boxer automatically completes the tag with `</center>` and places the text cursor between the tags.

Auto Tag Close file extensions

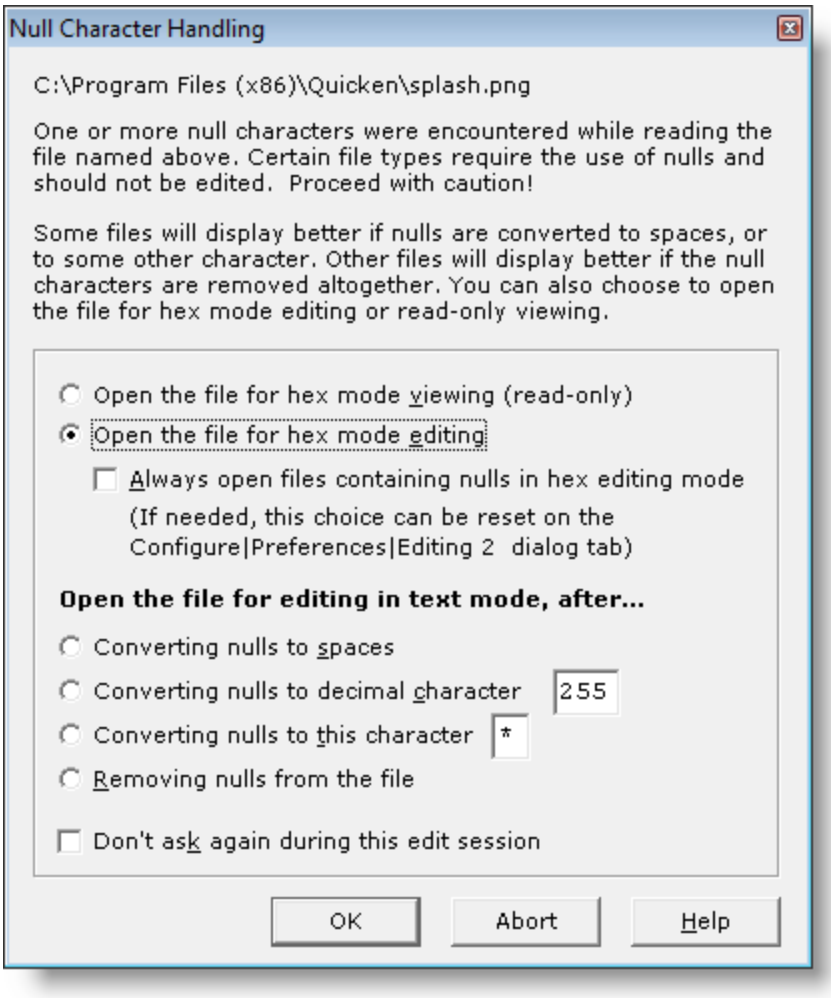
Use this option to control which file types are eligible for the Auto Tag Close feature. When the ellipsis (...) button is clicked, a small edit window appears which contains the eligible file extensions.

Automatically close parenthetical characters when typed
When this option is active, parenthetical characters will be automatically closed when they are typed from the keyboard. The text cursor is then placed between the mated characters. An edit box is provided to control which characters will be automatically closed. This option is off by default.

By necessity, if < and > are designated among the list of mating characters, the auto-close feature for HTML and XML tags will be disabled.

Editing options

Open files containing nulls in hex editing mode
Display a dialog with options for handling null characters
These options control how Boxer reacts when a request is made to open a file that contains null characters. If the first option is selected, Boxer will automatically open the file in hex editing mode. If the second option is selected, the Null Character Handling dialog will appear before the file is opened, providing additional options for how the file can be handled:



Word/Sentence/Title case commands will convert text to lowercase before operation

This option causes the Word, Sentence and Title case commands to automatically convert the selected text to lowercase before performing their function. When operating on uppercase text, this mode of operation allows the desired conversion to be performed in one step.

But take note: if the Word case command is applied to the following text:

```
IBM, eBay, MasterCard Report Record Profits
```

the result may not be as expected:

```
Ibm, Ebay, Mastercard Report Record Profits
```

Insert Symbols

The edit boxes within this section permit the definition of up to eight character values for use with the Insert Symbols feature. The values for the characters are entered in decimal format, and must reside in the range 1 to 255. The defined character is displayed to the right of each edit box using the same character set (ANSI or OEM) that is in use in the editor itself. This should help ensure that the characters are displayed as expected. If a character does not display within the dialog with the expected representation, this should not be cause for alarm. Simply verify that the character has the expected appearance when inserted into the text file.

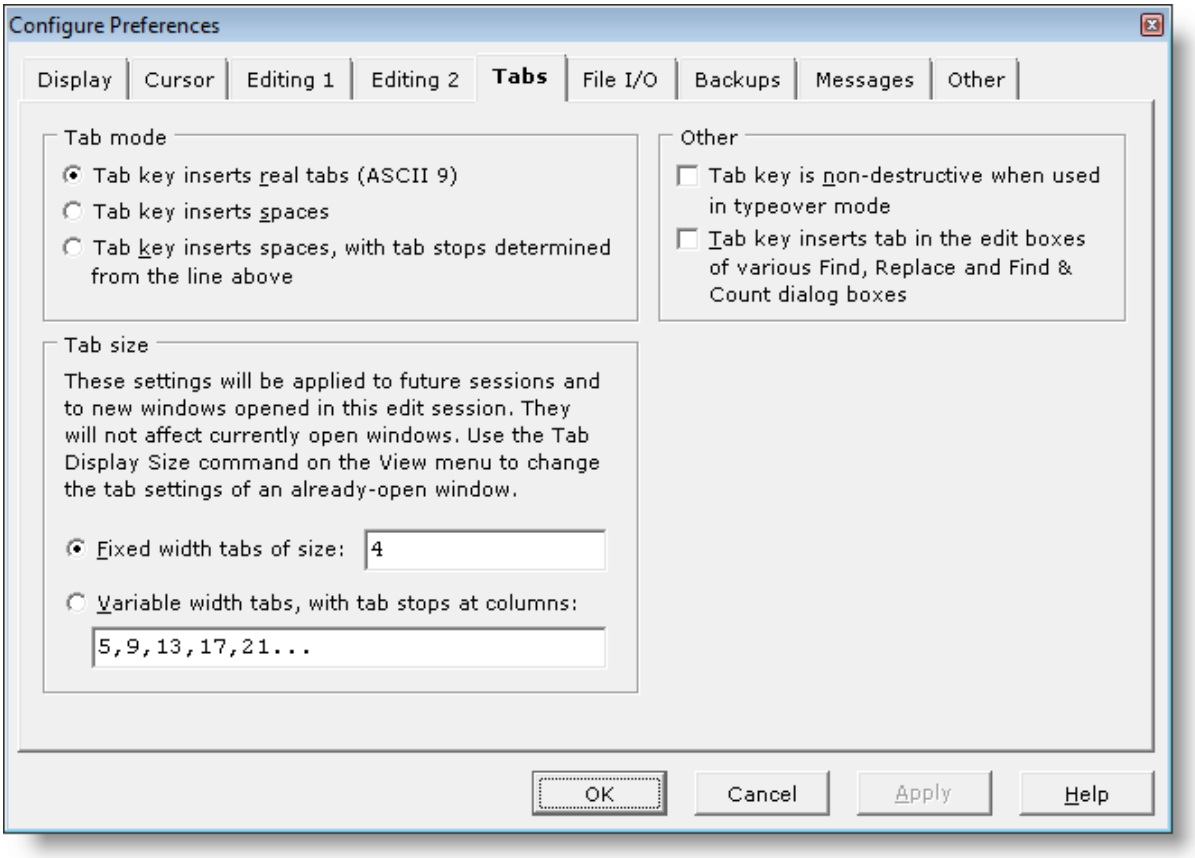
4.9.5 Preferences -Tabs

Menu: Configure > Preferences

Default Shortcut Key: none

Macro function: ConfigurePreferences()

The Tabs page of the Configure Preferences dialog contains options which relate to function of the *Tab* key, and to the display size of a tab:



Tab mode

Tab key inserts real tabs
When this option is used the *Tab* key will insert tabs (character value 9) into the file.

Tab key inserts spaces
When this option is used the *Tab* key will insert an equivalent number of Spaces, in accordance with the current Tab Display Size.

Tab key inserts spaces, with tabstops determined from line above
When this option is used, the *Tab* key will advance the text cursor to the next field of data as determined from the line above the current line.

Tab Size

Fixed with tabs of size *n*
Use this option to set the default display size for fixed width tabs.

This option sets the display size to be used for tabs in newly created windows, but it does not alter the tab display size of any editor windows that may already be open. The Tab Display Size command on the View menu can be used to set the tab display size for the current file, independent of this default value.

Variable width tabs, with tab stops at columns...

Use this option to designate the columns at which variable width tab stops should occur.


This option sets the tab stops to be used for tabs in newly created windows, but it does not alter the tab stop settings any editor windows that may already be open. The Tab Display Size command on the View menu can be used to set the tab stops for the current file, independent of this default value.

Other

Tab key is non-destructive when used in typeover mode
When this option is checked, the *Tab* key will not overwrite text when used in Typeover mode. For a similar option that affects the function of the *Backspace* key, see the Configure | Preferences | Tabs dialog page.

Tab key inserts tab in the edit boxes of various Find, Replace and Find & Count dialog boxes
When this option is checked, the *Tab* key with insert an actual tab character into the edit boxes of the Find, Replace, Replace Line Enders and Find & Count dialog boxes. Ordinarily, when the *Tab* key is pressed in a dialog box, focus shifts to the next control in the dialog box. This option can be used to override that behavior, making it easier to create search or replace strings that include the tab character.

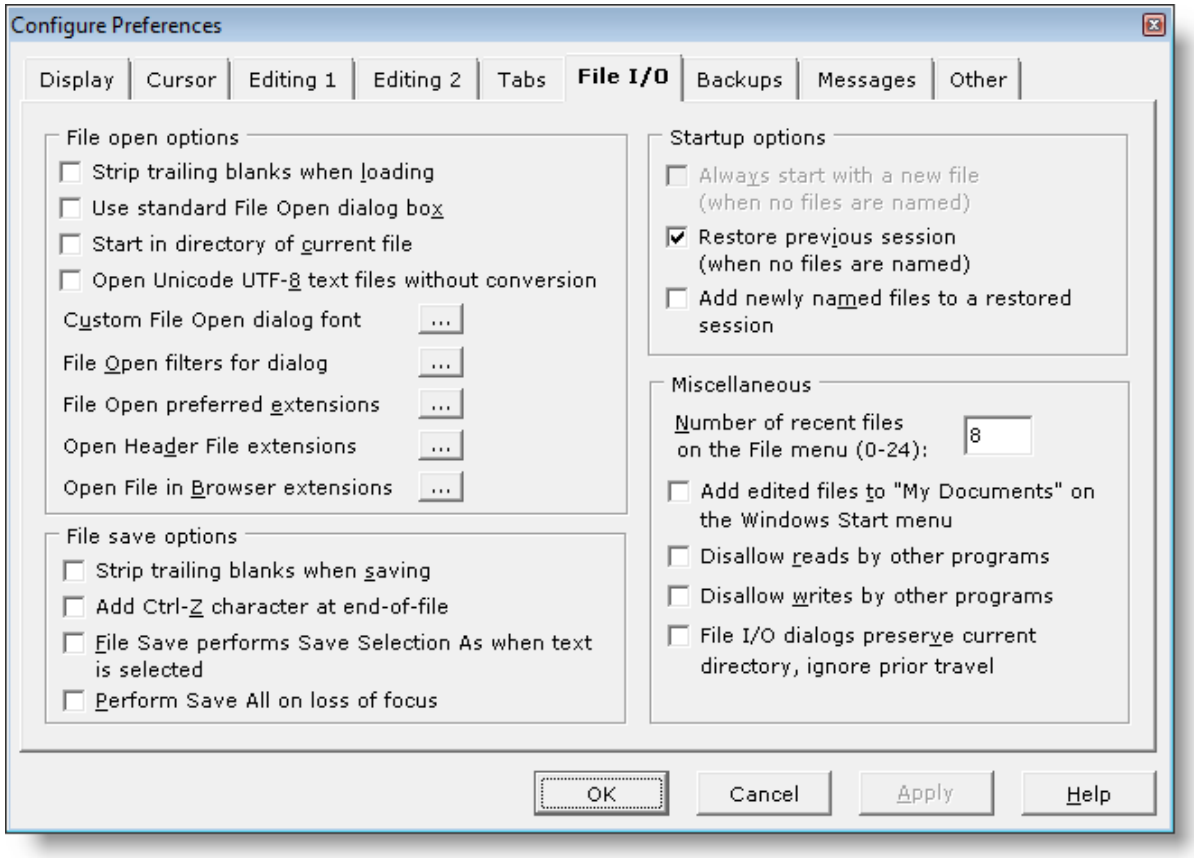
4.9.6 Preferences - File I/O

Menu: Configure > Preferences

Default Shortcut Key: none

Macro function: ConfigurePreferences()

The File I/O page of the Configure Preferences dialog box contains options which deal with the loading and saving of files:



File Open options

Strip trailing blanks when Loading a file
Use this option to request that trailing Spaces and/or Tabs be removed from the ends of lines as a file is loaded from disk. See also the option titled *Strip trailing blanks when saving a file*.

Use standard File Open dialog box
Use this option to specify that the standard Windows File Open dialog box should be used by the File | Open command. Boxer's custom dialog provides many features that are lacking in the standard dialog. Some users may be more comfortable with the standard dialog, and may wish to select this option.

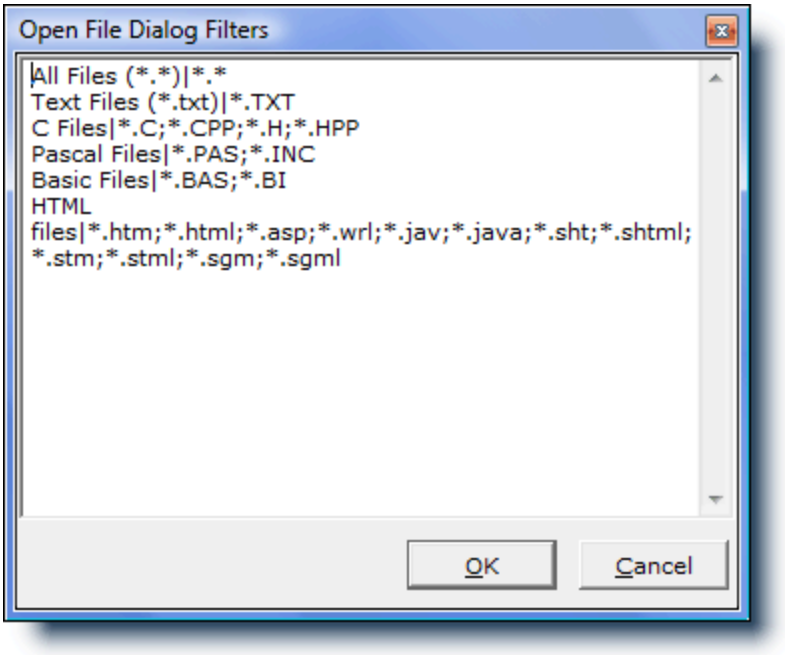
Start in directory of current file
When this option is selected, the File | Open dialog, File | Picker, and Find Text in Disk Files commands will start in the directory of the current file.

Custom File Open dialog font
Use this option to select the font that will be used in Boxer's custom File | Open dialog.

File Open filters for dialog
This option can be used to configure the list of *file filters* which are available from within the File | Open dialog box. File Open filters make it easy to display a group of files: the name of the group can be selected from the drop-down list of file types in the File Open

dialog box.

When the ellipsis (...) button is clicked, a popup editing window will appear which contains the currently defined filters:



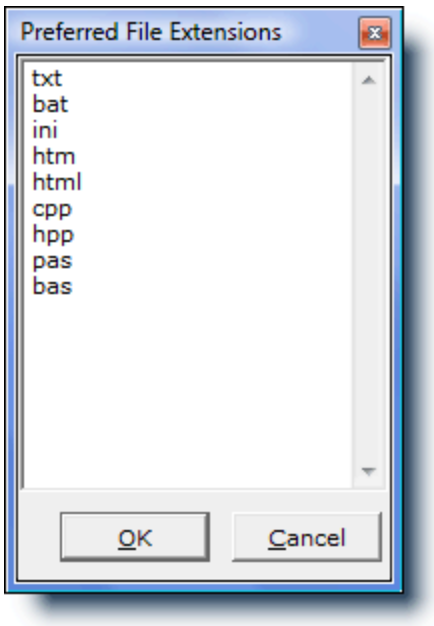
A file filter definition consists of two parts: the *filter name* and the *wildcard file specification*. The filter name contains the text string which will appear in the drop-down list of file types in the File Open dialog box. The wildcard file specification is the expression which is used to match the class of files being defined by the filter.

The filter name is separated from the wildcard file specification with the vertical rule (|) character. When more than one wildcard expression is used, the semi-colon (;) is used to separate the expressions.

File Open preferred extensions
This option can be used to configure Boxer's list of *preferred file extensions*. The list of preferred file extensions is consulted whenever Boxer is asked to open a filename which lacks a file extension.

Before opening a file each extension in the list is added to the supplied filename to see if a file by that name already exists. If so, the file is opened using the extension from the list. If no matching files can be found, the file is opened using the name as originally supplied. In a case where multiple matches might be found, the first matching extension will be used.

When the ellipsis (...) button is clicked, a popup editing window will appear which contains the currently defined extensions:

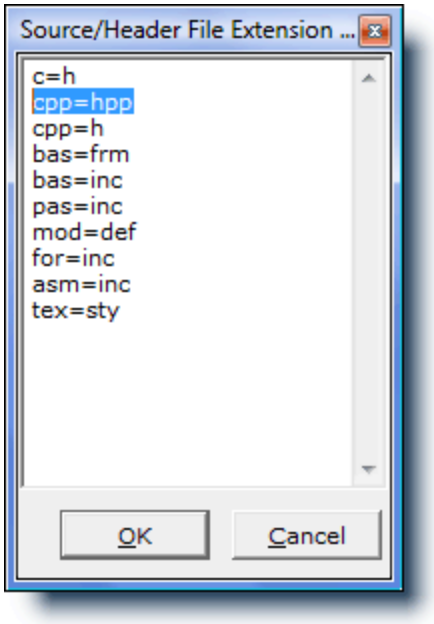


The list can be edited by entering desired extensions into the list, one-per-line. The period (.) should not be included in the extension.

☞ You may need to open a new file named TEST when TXT is a preferred extension, and the file TEST.TXT already exists. In this case, simply specify TEST., with the trailing period, in order to defeat the Preferred File Extension feature.

Open Header File extensions
This option can be used to configure the list of header file extension pairs which are used by the Open Header File command.

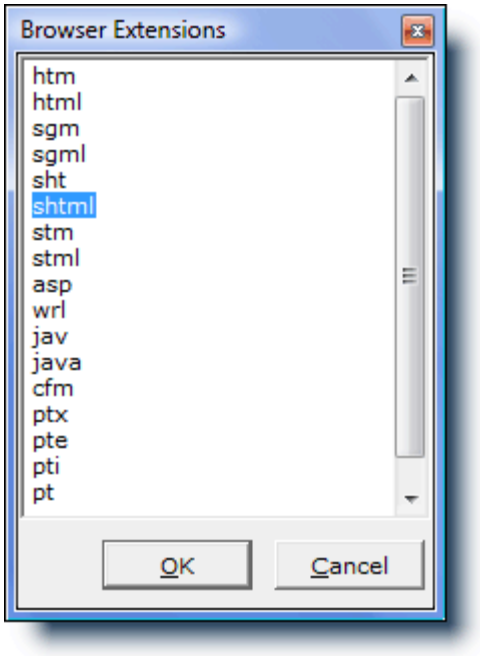
When the ellipsis (...) button is clicked, a popup editing window will appear which contains the currently defined pairs:



Extension pairs are listed one-per-line, with the equal sign (=) being used to separate the extensions. In a case where a file extension has multiple mates, the mate which occurs first in the list will be given priority when attempting to locate the file to be opened.

Open File in Browser extensions
This option can be used to configure the list of file extensions which is used by the Open File in Browser command to determine whether a file is eligible to be opened in an Internet browser.

When the ellipsis (...) button is clicked, a popup editing window will appear which contains the currently defined extensions:



Eligible file extensions are listed one per line. As the HTML standard changes, and as Internet browsers evolve, additional file types will likely become eligible for inclusion in the list.

File Save options

Strip trailing blanks when saving a file
Use this option to request that trailing Spaces and/or Tabs be removed from the ends of lines as a file is saved to disk. See also the option titled *Strip trailing blanks when loading a file*.

Add Ctrl-Z character at end-of-file
This option can be used to request that a Ctrl-Z character--ASCII 26, also known as the end-of-file (EOF) character--be added to a file when saving. Some older programs may require that a file be terminated in this way, but very few modern software packages do.

☞ This option is only applicable when the File Encoding format is set to ASCII; see File Properties for details.

File Save performs Save Selection As, when text is selected
When this option is active and a text selection is present, the Save command will perform the function of the Save Selection As command, saving the *selection* to a specified disk file rather than saving the *file* itself.

Perform Save All upon loss of focus
This option causes Boxer to perform the Save All command whenever focus shifts to another application.

Startup options

Always start with a new file (when no files are named)
Use this option if you prefer that Boxer open a new, untitled file whenever it is launched and another filename is not supplied. This option is not available when the *Restore previous sessions* option is active.

Restore previous sessions (when no files are named)
Use this option if you prefer that Boxer restore the previous edit session whenever it is launched and another filename is not supplied. The restored session will maintain the sizes and positions of all windows, the cursor position in each file, split windows status, and much more. This option is not available when the *Always start with a new file* option is active.

Add newly named files to a restored session
Use this option if you prefer that newly named files be *added* to the edit session which is being restored. This option is not available unless the *Restore previous sessions* option is also active.

Miscellaneous

Number of recent files on the File Menu (0-24)
Use this option to control the number of files which are displayed in the Recent Files list. Up to 24 files can be displayed in this list.


When using Boxer on screens with 800 x 600 resolution it will be necessary to set the number of recent files to four (4) or fewer to prevent the File menu from exceeding the screen height.

Add edited files to 'My Documents' on the Windows Start menu
Use this option to control whether or not files edited by Boxer are added to the *Documents* menu available from the Windows *Start* menu. The Documents menu is preferred by some users as a means to recall previously edited files.

Successful use of this technique requires that the file extension of the file being recalled is 'owned' by the application which last opened the file. This type of 'ownership' is achieved by the use of file associations. By its very nature, a text editor is likely to be called upon to edit many different file types (file extensions). It is probably *not* desirable for a text editor to own the file associations for all the file types it will be asked to edit. Therefore, using the Document menu to launch Boxer will be successful only for file types with which Boxer has been associated.

Disallow reads by other programs
Use this option to request that files which are opened for editing within Boxer be 'locked' so that they cannot be read by other programs. Use of this option will prevent a file from being viewed passively by another program so long as the file is open within Boxer.

Disallow writes by other programs
Use this option to request that files which are opened for editing within Boxer be 'locked' so that they cannot be written to by other programs. Use of this option will

prevent a file from being modified by another program so long as the file is open within Boxer.


A file which is being edited within Boxer could be modified by another program or process. If this condition occurs it will be reported by Boxer as soon as Boxer regains *focus*, and an option will be provided to reload the modified file from disk. An option to disable this option appears on the Configure | Preferences | Messages option page.

File I/O dialogs preserve current directory, ignore prior travel
This option can be used to prevent the various File I/O dialogs (Open, Save, etc.) from changing the record of the current directory due to any directory travel performed from within those dialogs. Ordinarily, the directory last visited within a dialog box is recorded so that it can be used when the dialog next becomes active. This option ensures that directory travel within a dialog box does ***not*** change the record of the current directory.

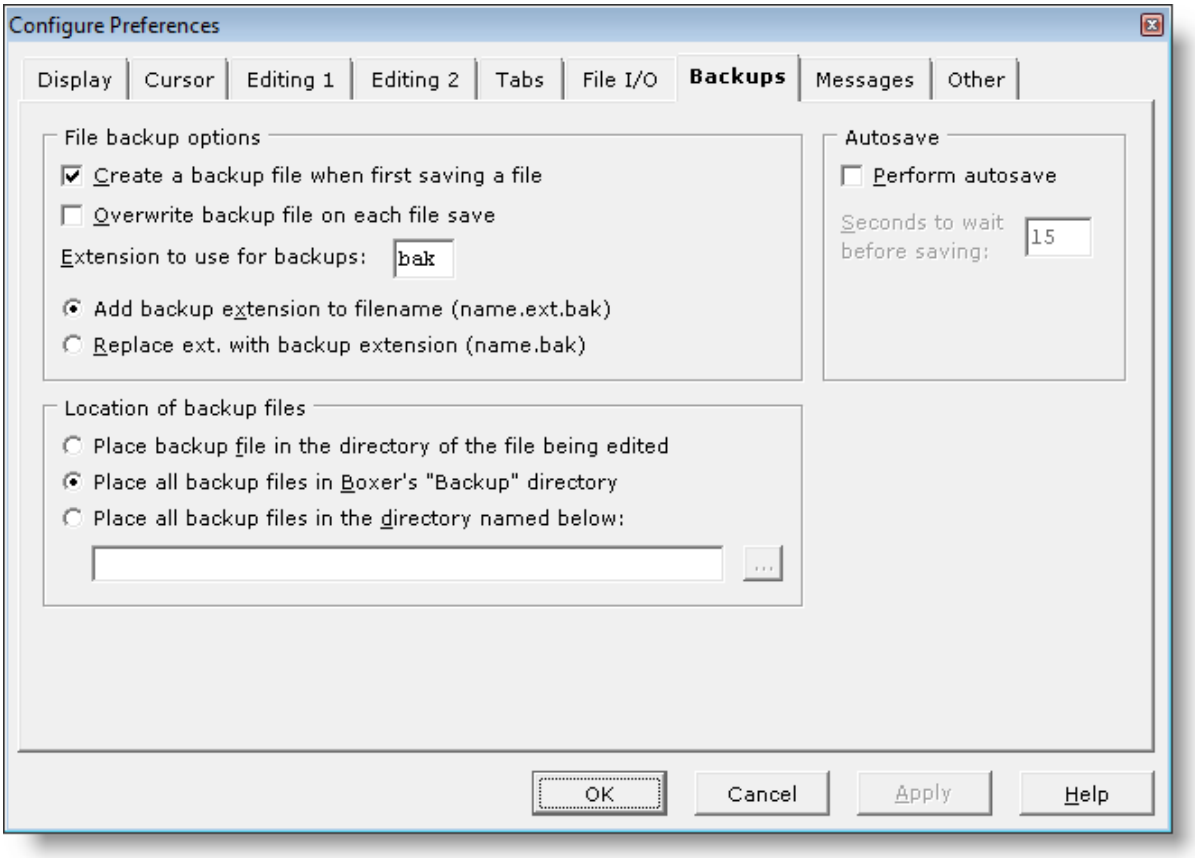
4.9.7 Preferences - Backups

Menu: Configure > Preferences

Default Shortcut Key: none

Macro function: ConfigurePreferences()

The Backups page of the Configure Preferences dialog box contains options which deal with file backups and the Autosave feature:



File backup Options

Create a backup file when first saving a file
Use this option to request that a backup file be created the first time a file is saved. Subsequent Save operations will not disturb the backup file. If this option is unchecked, all other backup options are disabled.

Overwrite backup file on each file save
Use this option to request that a new backup file be written every time File | Save is performed. When this option is not selected, the backup file will be written the first time a file is saved, but not thereafter.

Extension to use for backups
Use this option to specify the extension which is to be used for file backups. The extension can be 1 to 3 characters long.

Add backup extension to filename (name.ext.bak)
Use this option if you prefer that the backup file extension be *added* to the filename without removing any existing file extension.

Replace extension with backup extension (name.bak)
Use this option if you prefer that the backup file extension *replace* any existing file extension.

Location of backup files

Place backup file in the directory of the file being edited
Use this option to specify that backup files be placed in the same directory as the file being edited. The ellipsis (...) button can be used to browse for a directory using a standard dialog.

Place all backup files in Boxer's "Backup" directory
Use this option to specify that all backup files be placed in the "Backup" directory (folder) which appears in Boxer's home (installation) directory.

Place all backup files in the directory named below
Use this option to specify that all backup files be placed in the directory name which is provided in the associated edit box.


If you frequently edit files of the same name which exist in different directories, you may wish to choose that backup files be kept in the directory of the file being edited. Otherwise, if a common backup directory is used, it's possible that a backup file could be overwritten when later editing a file of the same name from within a different directory. For example, when editing the files `c:\east\sales.txt` and `c:\west\sales.txt`, the file which is saved second would be the one for which the backup file `c:\boxer\Backup\sales.txt.bak` applies.

Autosave

Perform Autosave
This option can be used to indicate that edited files be saved automatically after the supplied time interval has elapsed.


When editing a new, untitled file (see File | New), the Autosave feature will be not be active until the file is saved for the first time, and a filename has been assigned.


When using Autosave on a file that resides on a USB flash drive, it's worth considering the lifetime write limitation of the device. Some USB drives advertise that their lifetime write limitation runs between 10,000 and 1,000,000 uses, so if Autosave were used excessively, it could diminish the life of a USB drive.

Seconds to wait before saving

Use this option to specify the number of seconds after which Autosave should be performed. Caution: using a value which is too small could interfere with data entry when a very large file is being edited, or on very slow PCs.

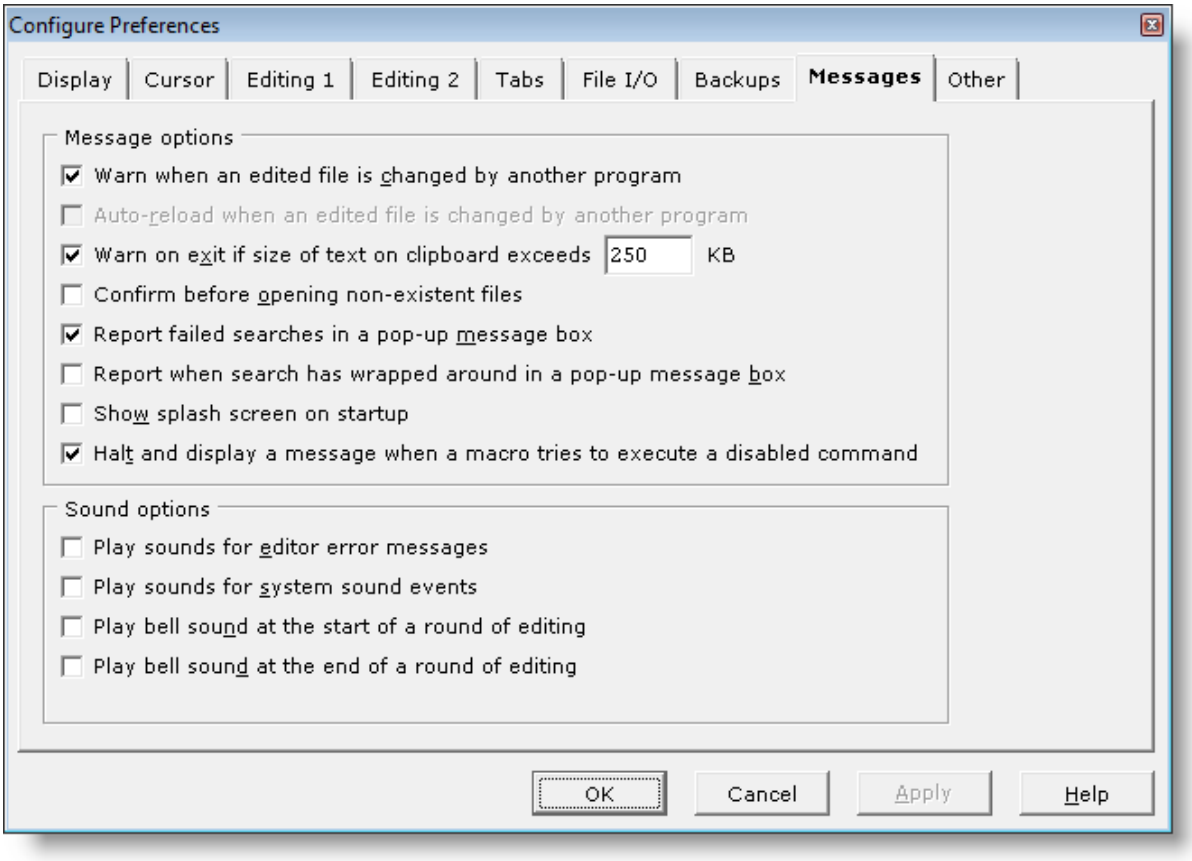
4.9.8 Preferences - Messages

Menu: Configure > Preferences

Default Shortcut Key: none

Macro function: ConfigurePreferences()

The Messages page of the Configure Preferences dialog box contains options which relate warning messages and sounds.



Message Options

Warn before loading binary files
Use this option if you prefer to be warned before a *binary file* is loaded. Binary files contain the Null character (value 0) and cannot be reliably edited with Boxer. You may wish to use the Open Hex Mode command to view such files in a hex format viewing mode. Boxer will automatically use Open Hex Mode when asked to open files with the extension `.COM`, `.EXE` or `.DLL`.

Warn when an edited file is changed by another program
Use this option if you would like to be notified by Boxer when one of the files you are editing is modified by another program or process. If this option is checked, a dialog box will appear offering a chance to reload the file from disk. If this option is unchecked, notice will NOT be provided when an edited file is modified by another process, and the file will NOT be reloaded..

☞ Proceed carefully in a situation such as this: changes you have made to the file may

be lost when you reload, or saving your file again could overwrite changes which were made by another user.

Auto-reload when an edited file is changed by another program
Use this option if you would like Boxer to automatically reload a file (without issuing a warning) when it senses that it has been changed by another program or process.

☞ Consider carefully the effect of this option: if changes have been made to a file within Boxer and have not been saved, they will be lost if the file is reloaded.

Warn on exit if size of text on clipboard exceeds *n* KB
This option is used to present a warning on exit when the size of the text on the clipboard exceeds the designated threshold. When excessive amounts of text are left on the clipboard, system performance can be impacted.

Confirm before opening non-existent files
This option controls how Boxer reacts when asked to open a file which does not yet exist. If this option is active, a dialog box will be presented to confirm that a new file is to be created. An option is provided to correct a typing error, in case that was the cause for the file not being found. If this option is inactive, Boxer will create a new file using the name provided.

Report failed searches in a pop-up message box
This option controls how Boxer will report a failed search. If this option is active, a popup dialog box will be used to report the failure. Otherwise a message will appear on the Status Bar.

Report when search has wrapped around in a pop-up message box
When Find Next or Find Previous are used in wrap around mode, a message appears on the status bar when the search has wrapped back to the location of the first match. Use this option if you prefer that this event be reported in a message box instead.

Show splash screen on startup
This option controls whether or not Boxer's splash screen graphic will be displayed on-screen while the program initializes. Display of the splash screen graphic can also be controlled using the -G command line option flag.

Halt and display a message when a macro tries to execute a disabled command
This option is intended for advanced users. By default, Boxer will prevent macro functions from being executed when the underlying command being run is disabled within the menus. Commands are disabled when the environment is unsuitable for them to be run. Advanced users may wish to override this behavior, if they think they have reason to disregard Boxer's disabling of a given command.

Sound Options

Play sounds for editor error messages
Use of this option causes the system sound for *Default Beep* to be played for editor errors.

Play sounds for system sound events

Use of this option causes the system sounds for *Minimize*, *Maximize*, *Restore Up* and *Restore Down* to be played when these events occur.

☞ The sounds which are used for various system sounds can be defined from Start Menu | Settings | Control Panel | Sounds.

Play bell sound at the start of a round of editing
Use of this option causes Boxer to play its two-bell sound to signal the beginning of a new round of editing.

Play bell sound at the end of a round of editing
Use of this option causes Boxer to play its one-bell sound to signal the end of a round of editing.

☞ If your computer is not equipped with a sound card, Boxer will not be able to play the sounds described above.

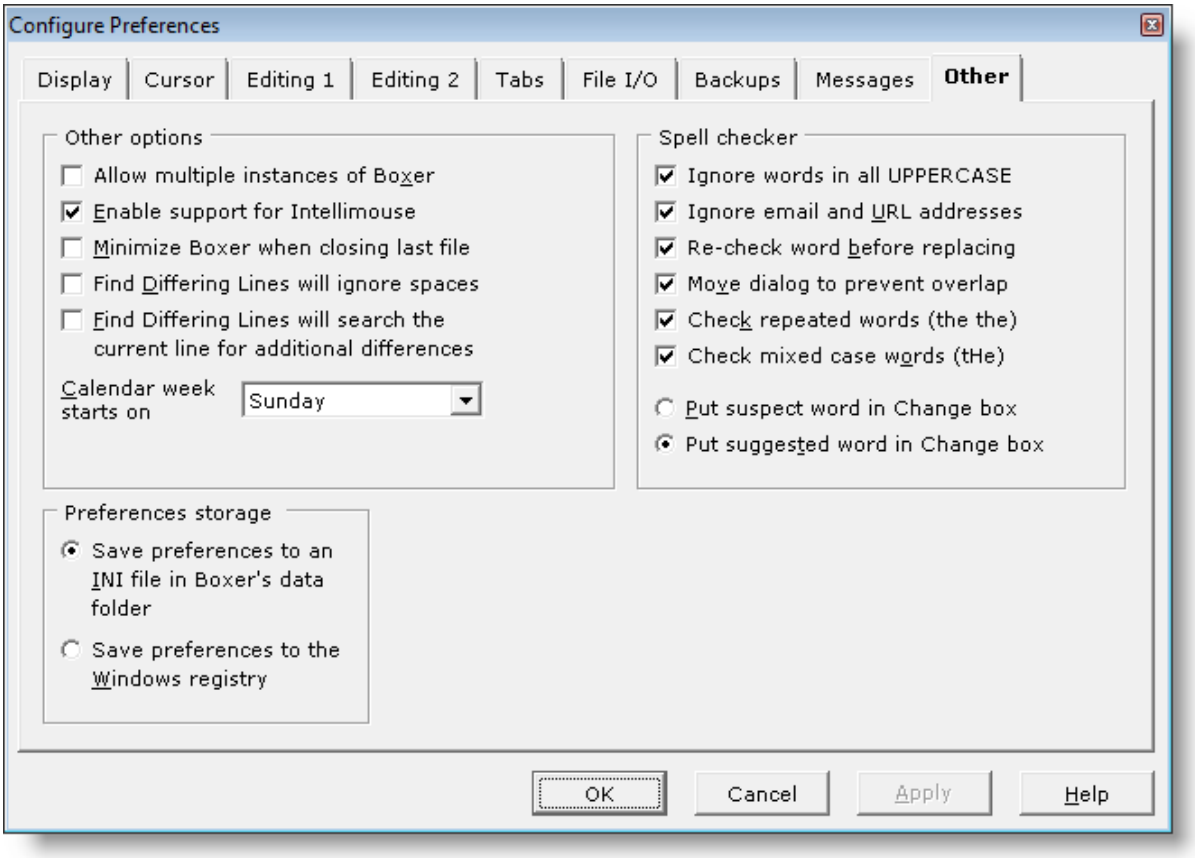
4.9.9 Preferences - Other

Menu: Configure > Preferences

Default Shortcut Key: none

Macro function: ConfigurePreferences()

The Other page of the Configure Preferences dialog box contains options which do not logically group with its other pages. Among these are startup options, Spell Checker options and miscellaneous options:



Other options

Allow multiple instances of Boxer
Use this option to control whether or not multiple instances of Boxer are allowed to run concurrently. When multiple instances are allowed, a new Boxer session will be launched each time the program is run. When multiple instances are disallowed, each would-be instance of Boxer will pass the files named for editing to the session which is already running. If files are not named for editing the existing session will simply become current.

☞ **Caution**: when multiple instances of Boxer are run concurrently confusion can arise regarding the editor's configuration settings. Boxer writes its configuration information to the *Windows registry* each time it exits. When multiple instances of Boxer are run the configuration settings from the first sessions that are exited will be overwritten by any sessions which exit later on. In essence, the configuration settings from the last exited Boxer session will be the settings which are ultimately recorded in the registry.

Enable support for Intellimouse
Use this option to enable support for the Microsoft Intellimouse device. The Intellimouse has a *mousewheel* which permits a document to be scrolled without the use of the Vertical Scroll Bar. The mousewheel also doubles as a center mouse button, and can therefore be used to perform columnar text selections.

Minimize Boxer when closing last file
When this option is active, Boxer will minimize itself to the *task bar* when its last file is closed. Click on the Boxer button in the task bar to restore the application.

Find Differing Lines will ignore spaces
Use this option to force the Find Differing Lines command to ignore leading and trailing Spaces and Tabs when comparing lines. Lines which differ only in their indent, or due to trailing *whitespace*, will be considered equal.

Find Differing Lines will search the current line for additional differences
Use this option to instruct the Find Differing Lines command to continue search along the current line for differences after the first difference has been reported. A manual re-syncing of the text cursor position may be required.

Word/Sentence/Title case commands will convert text to lowercase before operation
When applying Word, Sentence or Title case to a text selection, the question arises whether or not the text should first be forced to lowercase before the operation is performed. When this checkbox is checked, the case of the selected text *will* be adjusted before applying the requested conversion. You may wish to review the text after conversion to ensure that proper nouns, acronyms, and other capitalized words have been properly converted.

Calendar week starts on
Use this option to designate the day of week which should be used to start a week in the pop-up Calendar. The default setting is Sunday, but users in some countries will prefer a different setting.

Preferences Storage

Save preferences to an INI file in Boxer's data folder
When this option is selected, Boxer will save its preferences to a disk-based file named `BOXER.INI` which is located in its *data folder*. For more information, see Portable Editing.

Save preferences to the Windows registry
When this option is selected, Boxer will save its preferences to the Windows registry. The location used is:

```
HKEY_CURRENT_USER/Software/Boxer Software/Boxer Text Editor NN
```

where 'NN' represents the current major version number.

Spell Checker

Ignore words in all UPPERCASE
Use this option to indicate that the Spell Checker should ignore words which appear in uppercase during its operation. This helps prevent false reports when spell checking files which contain *acronyms* or filenames.

Ignore Email and URL addresses
Use this option to indicate that the Spell Checker should ignore email and *URL* addresses during its operation.

Recheck word before replacing
Use of this option causes the Spell Checker to recheck a replacement word before making the change. This provides a double check when you elect to type a replacement word rather than using one from the supplied list.

Move dialog to prevent overlap
Use this option to request that the Spell Checker position its dialog box so as not to obscure the context of the suspect word which is being reported. The position of the mouse cursor is moved along with the dialog, so you won't need to 'chase' the dialog around the screen throughout a spell checking session.

Check Repeated Words
Use this option to request that the Spell Checker watch for repeated words, such as 'the 'the'. When an offending sequence is found, an option will be provided to delete the duplicated word.

Check Mixed Case Words
Use this option to request that the Spell Checker check the spelling of mixed case words. Mixed case words can occur due to a typographical error, or when an acronym (GmbH) or company name (SoftSeek) is being used.


Due to a limitation in the way the dictionary vendor stores entries in its user dictionary, it is not possible to add a legitimate mixed case word (such as eBay) so that future alerts for that word will not occur.

Put suspect word in Change box
Use this option if you prefer that the suspect word be placed in the Change edit box.

Put suggested word in Change box
Use this option if you prefer that the suggested correction be placed in the Change edit box.

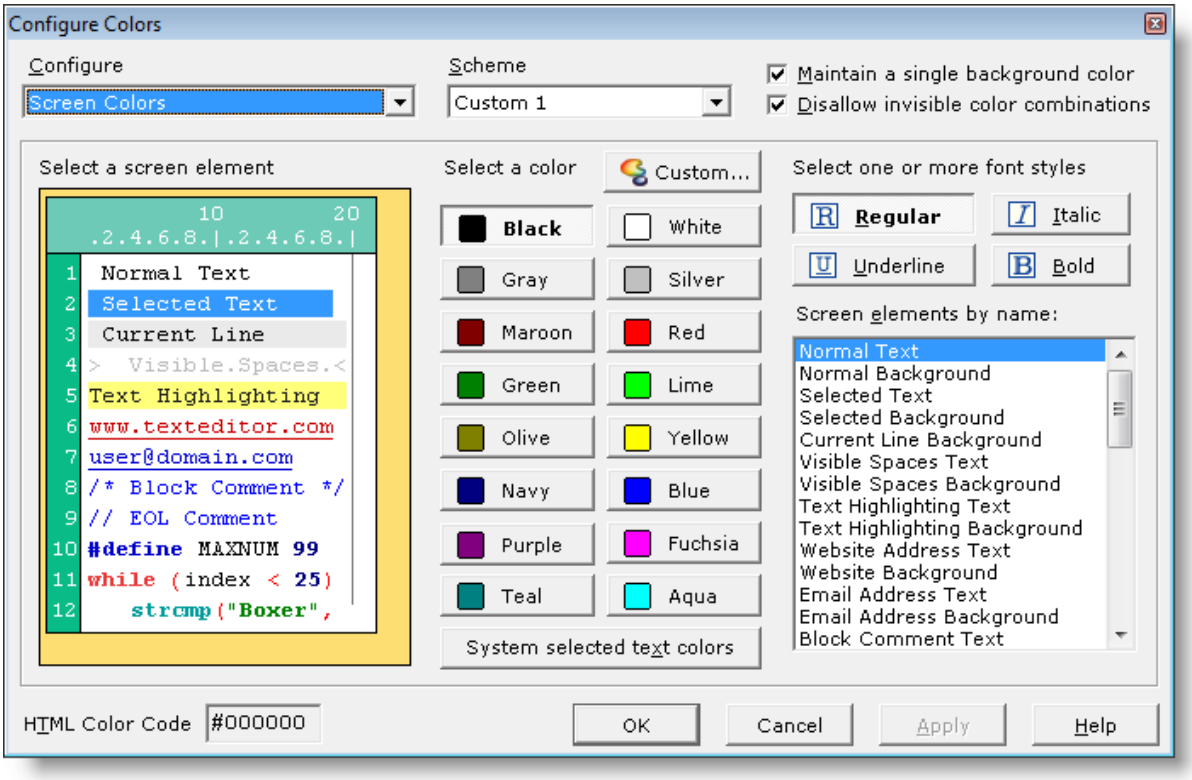
4.9.10 Colors

Menu: Configure > Colors

Default Shortcut Key: none

Macro function: ConfigureColors()

The Configure Colors command allows you to customize the colors that are used to display and print text files. Buttons are provided for the standard 16 colors. Use the *Custom* button to select from a palette of 16 million colors.



Configure
The Configure Colors dialog box operates in three different modes: *Screen Colors*, *Color Syntax Printing*, and *Monochrome Syntax Printing*. The active mode is selected from the *Configure* drop-down list at the upper left of the dialog box. *Screen Colors* mode allows you to set the colors used to display text files on-screen. *Color Syntax Printing* mode is used to set the colors used for printing program files on color printers. *Monochrome Syntax Printing* mode is used to set the colors used for printing program files on non-color printers.

Scheme
A set of pre-defined color schemes has been provided to speed the process of color configuration. You can start with the pre-defined scheme that is closest to your liking, and then make other changes as desired. Once a change has been made to a pre-defined color scheme you will be asked where to save the custom layout; four custom positions are available.

Maintain a single background color
This option can be used to ensure that all elements will use a single background color. When selected, Boxer ensures that all elements are updated when the background color is changed. Turn this option off if you would like to create a color scheme in which some elements use different background colors.

Disallow invisible color combinations
This option can be used to prevent any color selection which would cause the foreground and background colors of one or more elements to be the same.

System selected text colors
This button can be used to quickly apply the default text selection colors of the operating system to the current color scheme.


This option will be of particular use to blind users who are using Boxer with a screen reader such as JAWS. Screen reader software sometimes requires that the text selection colors used by an application match those that are used system wide.

HTML Color Code
As a convenience, the current color is displayed in HTML Color Code format to make it easy to duplicate a color used in Boxer in your HTML code. The HTML Color Chart command can also be used for selecting colors and getting an HTML Color Code value.

The process of changing colors is quite simple, and includes three steps:

1. Click to select an element
Click with the left mouse button in the miniature screen display on the element which is to be changed. You can click on the text of an element to select its foreground element, or on the background of an element to select its background element. After clicking, you'll see that the *Elements by name* listbox is updated to reflect the selected element. You'll also see that the color and font style buttons will be displayed in a depressed state to reflect the current settings for the selected element. If the element uses a color other than those appearing on the standard buttons, the *Custom* button will appear depressed. Some elements, such as the Right Margin Vertical Rule, cannot be easily selected by mouse in the miniature screen display. These elements can be selected from the *Elements by name* listbox instead.

2. Click to select a color
Click on the new color for the selected element. The miniature screen display will be updated to reflect the new color. When configuring for Monochrome Syntax Printing, the available colors will be reduced to those which can be achieved on non-color printers.

3. Click to select font style(s)
Click on one or more font styles for the selected element. The miniature screen display will be updated to reflect the new style. To remove a font style, click again on its button to clear the style.


When configuring screen colors, the *Apply* button can be used to update the screen below the dialog box to reflect the changes made.

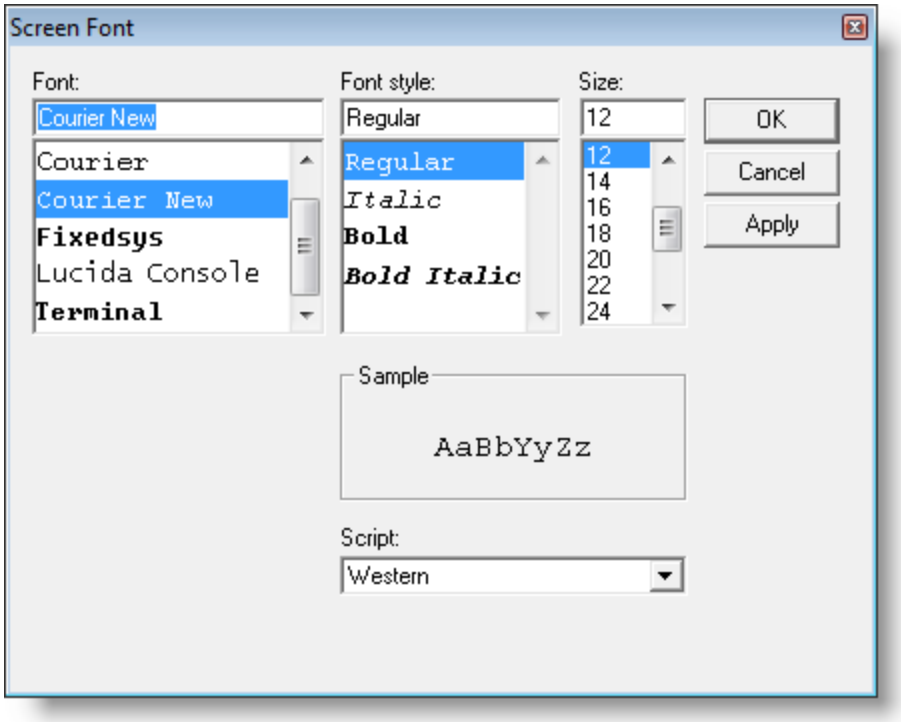
4.9.11 Screen Font

Menu: Configure > Screen Font

Default Shortcut Key: none

Macro function: ConfigureScreenFont()

The Screen Font command is used to select the font that is used to display edited files on-screen. The standard Windows font dialog is presented for selecting the screen font:



Boxer requires that *fixed width* fonts be used, so the Screen Font dialog box does not display *proportionally spaced* fonts. This is required, in part, to ensure that columnar selections can be highlighted neatly in rectangular blocks, and so that the Column Ruler can be used. These features would not be possible if the use of proportional fonts were permitted.

The *Font* listbox at the left of the dialog displays the *fixed width* fonts which are available for selection.

The *Font Style* listbox display the styles which are available for the selected font. Typically these are *Regular*, *Italic*, *Bold* and *Bold Italic*, although some fonts may not offer all styles.

The *Size* listbox displays the sizes which are available for the selected font.

The *Script* drop-down list displays the various character mappings which are available for the selected font.

The color that will be used to display the font selected is controlled via the Configure Colors command.

Tips and Notes

You may have need to display files which were created using a DOS program and

which contain characters from the upper half of the ASCII character set. To display these files properly select a font which offers an *OEM/DOS* script style. One such font that is available on most systems is the *Terminal* font.


At the time of this writing, the Internet site http://keithdevens.com/wiki/ProgrammerFonts contained good information about fixed width fonts, along with links to several screen fonts which could be downloaded free of charge. The Dina Programming Font is a very nice, free, fixed width font.

Changing the *Font Style* selection from this dialog will cause the font style of 'Normal' text to be changed to the selected style. It will not alter the font styles of other screen syntax elements such as Comments, Reserved Words, Strings, etc. The Configure Colors command can be used to select the font style(s) for these elements, as well as for Normal text. For maximum flexibility it may be advisable to let the font style remain as 'Regular' in the Screen Font dialog and select the font styles to be used for Color Syntax Highlighting from the Configure | Colors dialog.


The Screen Font is not used to print files; see the Printer Font command to select a font for that purpose.


When selecting a True Type font, the standard Windows font dialog box may display a message at the bottom indicating that the selected font will be used for both screen display and printing. This message does *not* apply to the use of fonts within Boxer, since Boxer permits the Screen Font and Printer Font to be selected separately. The message intends to convey the idea that the font is *capable* of being used for both the screen and the printer.

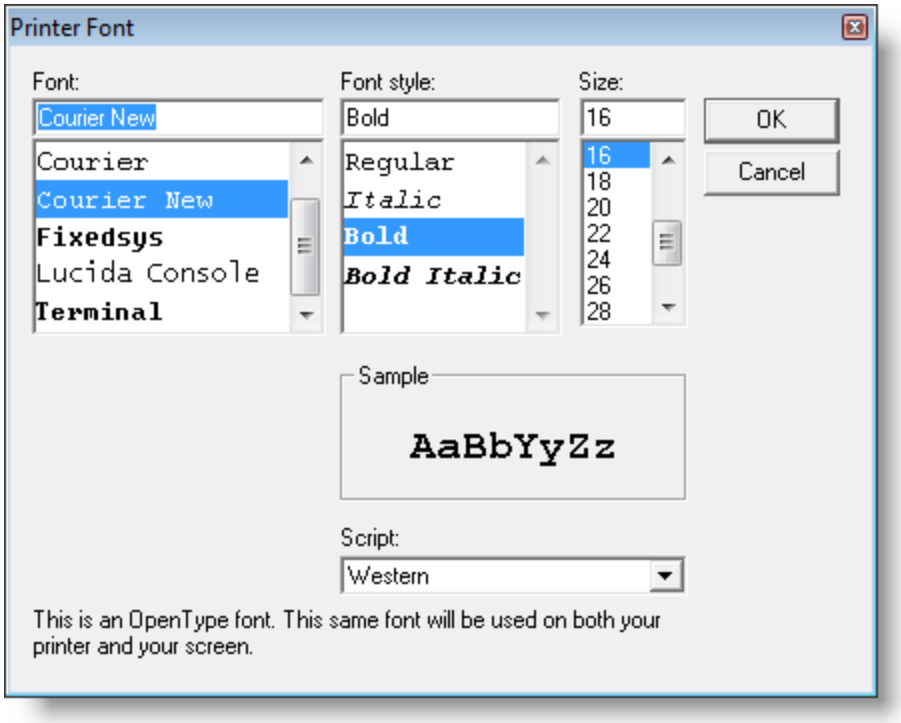
4.9.12 Printer Font

Menu: Configure > Printer Font

Default Shortcut Key: none

Macro function: ConfigurePrinterFont()

The Printer Font command is used to select the font that is used to print files from within Boxer. The standard Windows font dialog is presented for selecting the screen font:



☞ Boxer requires that *fixed width* fonts be used, so the Printer Font dialog box does not display *proportionally spaced* fonts. This is required, in part, to ensure that columnar selections can be highlighted neatly in rectangular blocks, and so that the Column Ruler can be used. These features would not be possible if the use of proportional fonts were permitted.

The *Font* listbox at the left of the dialog displays the *fixed width* fonts which are available for selection.

The *Font Style* listbox display the styles which are available for the selected font. Typically these are *Regular*, *Italic*, *Bold* and *Bold Italic*, although some fonts may not offer all styles.

The *Size* listbox displays the sizes which are available for the selected font.

The *Script* drop-down list displays the various character mappings which are available for the selected font.

The colors which are used when using Color Syntax Printing are controlled via the Configure Colors command.

☞ When selecting a True Type font, the standard Windows font dialog box may display a message at the bottom indicating that the selected font will be used for both screen display and printing. This message does *not* apply to the use of fonts within Boxer, since Boxer permits the Screen Font and Printer Font to be selected separately. The message intends to convey the idea that the font is *capable* of being used for both the screen and the printer.

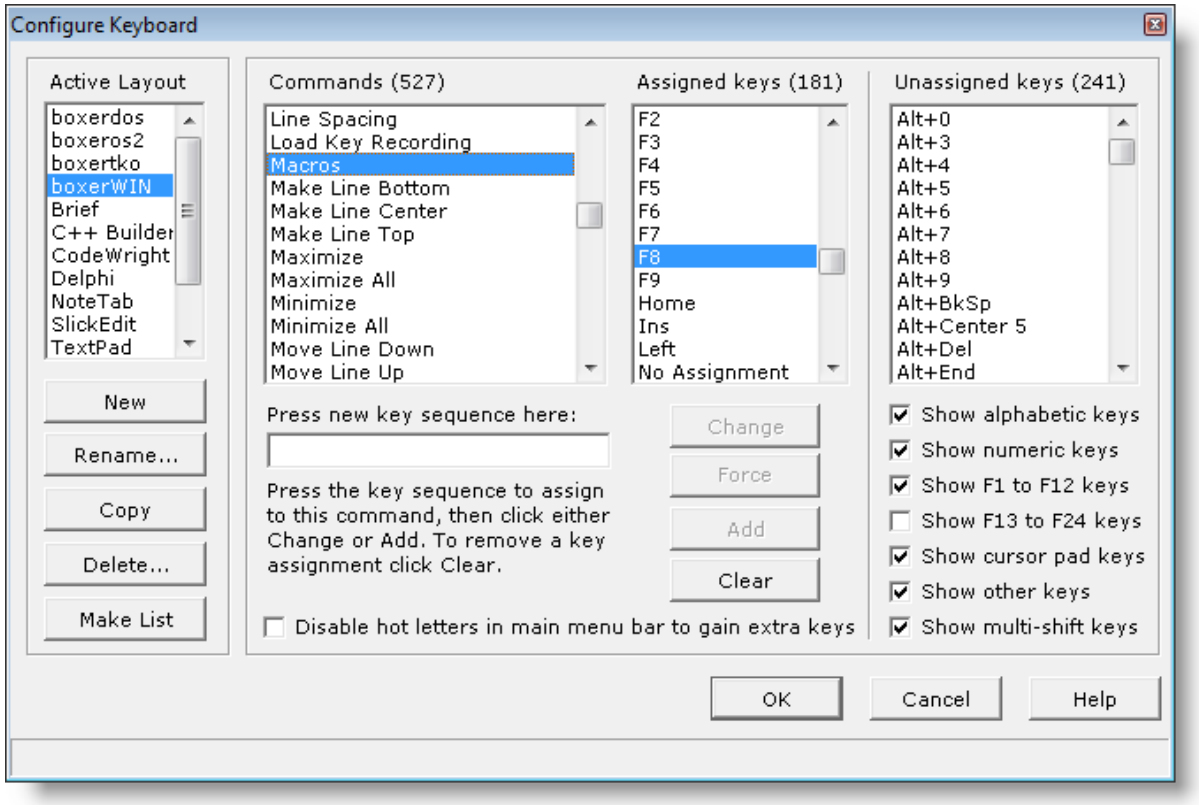
4.9.13 Keyboard

Menu: Configure > Keyboard

Default Shortcut Key: none

Macro function: ConfigureKeyboard()

The Configure Keyboard command provides the ability to assign key sequences to any of Boxer's commands. With over 450 editor commands, and over 400 key sequences, it's easy to think that keyboard configuration might be a complex undertaking. Not so. Boxer's Configure Keyboard dialog automates the process by providing lists from which *Commands*, *Assigned keys* and *Unassigned keys* can be selected, and by allowing key assignments to be typed directly from the keyboard.



A thorough coverage of the features of the Configure Keyboard dialog box is presented further below. In the paragraphs that immediately follow, a sample configuration session is presented which illustrates how several common changes can be made to the default keyboard layout.

Goal
Create a new keyboard layout which duplicates the default layout, but with a few selected changes.

Discussion
Rather than making changes to a pre-defined keyboard layout, it's always advisable to create a new layout with the *Copy* or *New* button. Future upgrades to Boxer will overwrite the default keyboard layout `BOXERWIN.KBD` and other pre-defined layouts. Custom changes should **not** be made to these files. The *New* button will create a nearly empty layout, but that's not what we want now. Instead, click *Copy* to create a copy of the active keyboard layout, and then click *Rename* to provide the new layout with a name of your choice. You might choose to simply use your first name.

Change One
The Auto-Number command has no key assignment in the default keyboard layout, and you'd like it to have one. Click on the Auto-Number entry in the *Commands* listbox. The *Assigned keys* listbox is updated to show 'No Assignment'. Find a suitable key sequence in the *Unassigned keys* listbox, and click on it. The name of the key sequence selected appears in the edit box beneath the *Commands* listbox. Click the *Change* button to change the assignment for Auto-Number from 'No Assignment' to the selected key sequence.

Change Two
The Align Right command is assigned to *Ctrl+F9*, but you'd like to use that key for another command instead. Click on the Align Right entry in the *Commands* listbox. Click on the *Clear* button to relieve the command of its key assignment. The *Assigned keys* listbox is updated to show 'No Assignment', and the *Ctrl+F9* key is added to the *Unassigned keys* listbox.

Change Three
The Calculator command is assigned to *F11*, but you'd like it to also be available using the *Ctrl+F9* sequence, which was just freed by Change Two above. Click on the Calculator entry in the *Commands* listbox. Its current assignment of *F11* is displayed in the *Assigned keys* listbox. Click in the edit box beneath the *Commands* listbox to give it focus. Press the *Ctrl+F9* key sequence from the keyboard, and watch its name appear in the edit box. (Whenever you prefer, a key can be pressed in the edit box as an alternative to locating it in the *Unassigned keys* listbox.) Finally, click the *Add* button to create this additional assignment for the Calculator command. A duplicate entry is created for Calculator in the *Commands* listbox, reflecting the fact that there are now two distinct key assignments for this command.

Active Layout listbox

The *Active Layout* listbox displays a list of the available keyboard layouts, and highlights the active layout. Boxer comes with several pre-defined layouts which can make Boxer more closely match the key assignments of another editor or word processor.

If you develop a keyboard layout that matches the key assignments of another popular program, please consider sending it to us at info@boxersoftware.com so that we can make it available to other Boxer users. Keyboard layout files are kept in Boxer's *data folder*, and are given a `.KBD` file extension.

New button

Use the *New* button to create a new keyboard layout. The new layout will contain only the most fundamental key assignments, such as *Up*, *Down*, *Left*, *Right*, etc. The new layout is created with the name 'New'; use the *Rename* button to supply the name of your choice.

Rename button
Use the *Rename* button to change the name of the selected layout to a name of your choice.

Copy button
Use the *Copy* button to make a copy of the active keyboard layout. The new layout will be given the name 'Copy of', prefixed to the name of the active layout. Use the *Rename* button to supply a new name, if desired. Use of the *Copy* button is recommended when you will be making a small number of changes to an existing layout.

Delete button
Use the *Delete* button to delete the selected keyboard layout. A confirmation is required before the layout will be deleted. Once a layout is deleted it **cannot** be recovered, even if *Cancel* is later selected.

Make List button
The *Make List* button creates a file in a new editor window which lists all of the command key assignments in the selected layout. This file could be printed to create a command chart, or saved to disk for later reference.

Commands listbox

The *Command* listbox displays an alphabetical list of all commands which can be reassigned. Clicking on an entry in the *Commands* listbox displays its current assignment in the *Assigned keys* listbox. When the listbox has focus, pressing the first letter of a command will jump the selection bar to that command.

When a command has multiple key assignments, an entry will appear in the *Commands* listbox for each such assignment.

The number of commands displayed in the listbox is shown in parentheses at the top of the list.

Assigned Keys listbox

The *Assigned Keys* listbox displays an alphabetical list of all key sequences which are currently in use. Clicking on an entry in the *Assigned Keys* listbox displays the associated command in the *Commands* listbox. The 'No Assignment' entry does not normally map to a single command, and therefore will not display its associations.

When the listbox has focus, pressing the first letter of a command will jump the selection bar to that command.

The number of key sequences displayed in the listbox is shown in parentheses at the top of the list.

Type new key in this box

This is the edit box where a new key sequence is entered. The edit box can be filled by clicking on an available key sequence from the *Unassigned Keys* listbox, or by pressing a key sequence from the keyboard while the edit box has focus. When a key sequence is entered, its disposition is reported in a message just above the edit box. It might be reported as available, not available, in use, or as being used by the System.

Change button
The *Change* button is used to change the key assignment for the currently selected command to the key sequence displayed in the edit box. The *Change* button will remain disabled until a key which is eligible for assignment has been entered into the edit box.

Force button
The *Force* button is used to change the key assignment for the currently selected command *and* simultaneously remove its assignment from another command.

Add button
The *Add* button is used to create an additional assignment for the selected command. The *Add* button will remain disabled until a key which is eligible for assignment has been entered into the edit box. There is no limit to the number of duplicate key assignments that a command may have.

Clear button
The *Clear* button is used to release a key assignment from the currently selected command. The *Clear* button will be disabled when the current command has no assignment.

Disable hot letters in main menu bar to gain extra keys
This option can be used gain access to the *Alt+letter* key sequences which would otherwise be used to activate the main menu entries. When this option is selected the key sequences *Alt+F*, *Alt+E*, *Alt+B*, etc. become available for assignment to other commands. When the Configure Keyboard dialog is dismissed, the main menu will be redrawn without its hot letters underlined.

Alt+letter sequences which are not otherwise assigned will remain assigned to their respective menus. For example: if this option is selected, but *Alt+F* is not otherwise assigned, it will remain as the key assignment for dropping the File menu.

Regardless of the state of this option, the main menu hot letters will remain functional when the main menu has been activated by tapping the *Alt* key.


When loading a keyboard layout file, Boxer will look for key assignments which conflict with the main menu hot letters in order to determine if this option needs to be checked. If you select this option, but fail to assign any of the *Alt+letter* sequences to other commands, Boxer will sense this when the layout is next loaded, and the option will revert to unchecked. Conversely, if you load a layout which contains one or more key assignments which conflict with the main menu hot letters, Boxer will force this option to checked.

Unassigned Keys listbox

The *Unassigned Keys* listbox displays those key sequences which are available for assignment. Clicking on a key within this listbox causes the key to be displayed in the edit box beneath the *Commands* listbox.

The keys which are to be shown in the listbox can be controlled with various checkboxes:

Show alphabetic keys
Use this option to include the *A-Z* keys, in all their various shift states.

Show numeric keys
Use this option to include the *0-9* keys, in all their various shift states.

Show F1 to F12 keys
Use this option to include the *F1-F12* keys, in all their various shift states.

Show F13 to F24 keys
Use this option to include the *F13-F24* keys, in all their various shift states. Some new keyboards are now offering these additional functional keys.

Show cursor pad keys
Use this option to include the keys from the cursor motion pad, in all their various shift states.

Show other keys
Use this option to include keys which do not group into the categories above.

Show multi-shift keys
Use this option to control whether or not key sequences with multiple shifts should appear in the list.

Notes


You might notice that the controls in this dialog box do not have hot letters, as do other dialog boxes. This is because the edit box into which key sequences are typed must be able to receive all possible key sequences without losing focus to another control.


Assigning a key sequence to run a macro is a two-step process. One step is to make the desired assignment to the *Run Macro n* command using the Configure Keyboard dialog. The other step is ensure that the macro itself is numbered accordingly. See the Macros topic for further information.


It is not possible to use multi-key sequences, such as *Ctrl+K, Ctrl+B* in a key assignment.


When a given command has more than one key assignment it will have multiple

entries in the *Commands* listbox. The first assignment that appears in the *Commands* listbox is called the *primary command assignment*. Additional entries for that command are referred to as *secondary command assignments.* The key sequence associated with the primary command assignment is the one which will be displayed in the main menu next to the command.

☞ When an *Alt+Letter* sequence is used as a secondary command assignment, you will likely notice a beep when that key sequence is pressed. The beep occurs because the *Alt+Letter* sequence does not map to an underlined hot letter on the main menu bar, and it does not appear as a *shortcut key* in any of the main menu entries. The beep can be silenced by making the *Alt+Letter* sequence the primary command assignment, and letting the existing assignment become the secondary assignment. To do this, the existing primary assignment must be cleared, so the *Alt+Letter* assignment becomes the primary assignment. Then, the other assignment can be added back as the secondary assignment. The only effective difference between a primary assignment and a secondary assignment is that the primary assignment is displayed in the main menu.

☞ Because they appear in the main menu, primary command assignments are available whenever Boxer is running. Secondary command assignments do not appear in the main menu, and are therefore available only when a child editor window is open. Since most commands are meant to operate on text, this rarely poses a problem. But there are instances where trouble can arise. For example: assume that Ctrl+N is the primary command assignment for the File | New command, and Shift+Alt+N is its secondary assignment. If all child editor windows are closed, the Shift+Alt+N assignment will be non-functional. The primary assignment, Ctrl+N, would need to be used.

☞ The Configure Keyboard dialog recognizes the numeric keypad keys as distinct keys in all of their shifted and unshifted states. These keys appear in the *Unassigned keys* listbox as *Keypad 1*, *Keypad 2*, etc.

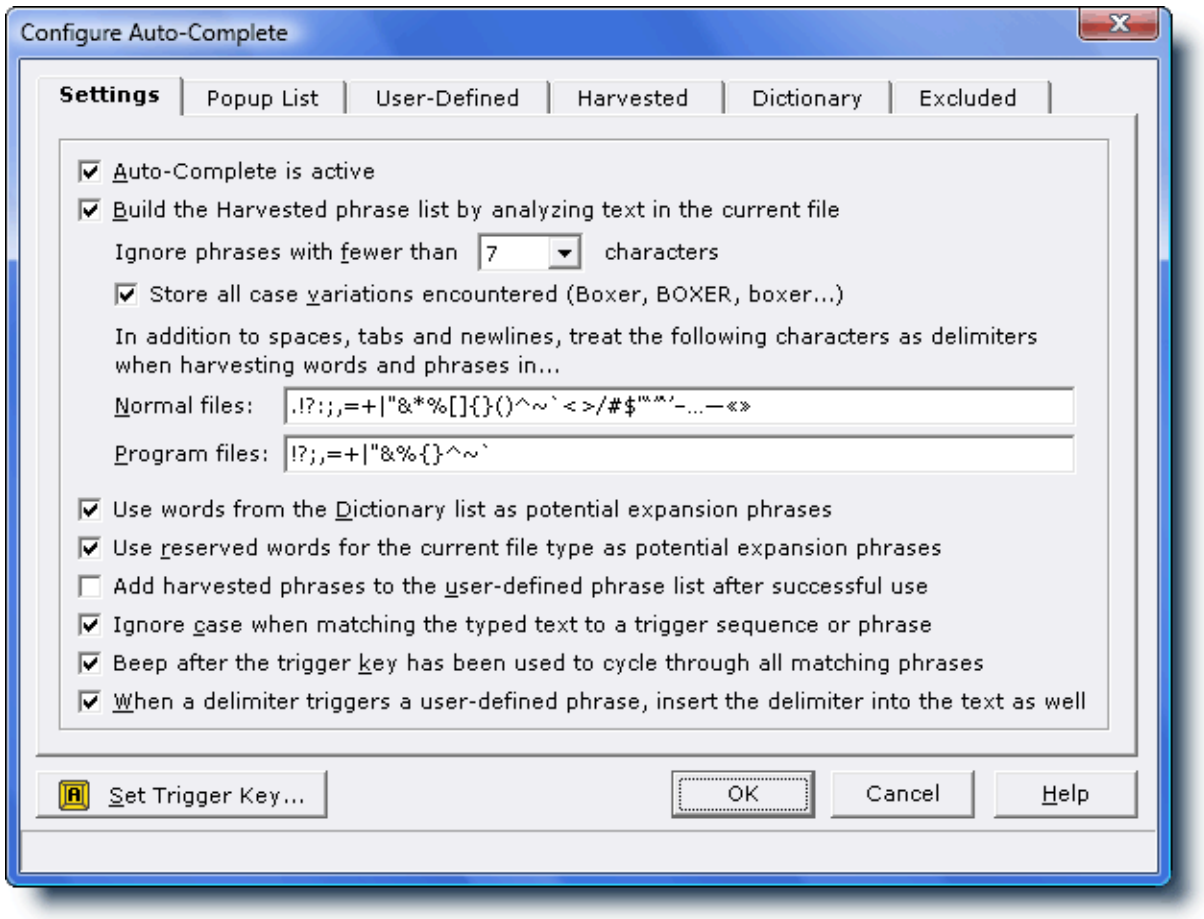
4.9.14 Auto-Complete - Settings

Menu: Configure > Auto-Complete

Default Shortcut Key: none

Macro function: none

The Settings tab of the Configure Auto-Complete dialog contains general options that relate to the Auto-Complete feature.



Auto-Complete is active
The checkbox is the master on/off switch for the Auto-Complete feature. If this box is unchecked, all Auto-Complete features and functions will be disabled.

Build the Harvested phrase list by analyzing text in the current file
Use this checkbox to control whether or not the current file will be analyzed to harvest words for use with the Auto-Complete feature. If harvesting is not performed, the matching word list will be built from other sources: user-defined phrases, reserved words, and the master dictionary.

Ignore phrases with fewer than *n* characters
Use this option to specify the shortest word that should be collected during harvesting. Auto-Complete is most effective when used to complete longer words, so there's little benefit to storing very small words when harvesting.

Store all case variations encountered (Boxer, BOXER, boxer...)
Use this checkbox to specify how case variations should be handled during harvesting. When checked, all different case variations will be collected. When unchecked, only the first case variation of a given word will be stored.

In addition to spaces, tabs and newlines, treat the following characters as delimiters when harvesting words and phrases in...

Normal files
These characters will be treated as word delimiters when harvesting words from normal files -- that is, from files that do not have a Syntax Highlighting definition entry.

Program files
These characters will be treated as word delimiters when harvesting words from program files -- that is, from files that **do** have a Syntax Highlighting definition entry.

The *Program files* delimiter characters control how the Auto-Complete feature harvests code fragments from the current file. This list of delimiters will typically have fewer characters in it than the *Normal files* list of delimiters, so that longer code fragments can be collected. Regardless of the delimiters listed here, Auto-Complete will also harvest program files with a more restrictive delimiter list so that the individual elements of a code fragment appear on their own. For example, the code fragment `structure_name->variable_name[index_variable]` might be collected in its entirety due to the *Program files* delimiters that are in use. *Regardless* of those delimiters, `structure_name`, `variable_name` and `index_variable` will be harvested as individual entries as well.

Use words from the Dictionary list as potential expansion phrases
This checkbox controls whether or not dictionary words will be used to build the matching word list.

Use reserved words for the current file type as potential expansion phrases
This checkbox controls whether or not reserved words from the Syntax Highlighting information for the current file will be used to build the matching word list.

Add harvested phrases to the user-defined phrase list after successful use
When this option is checked, harvested words will be automatically added to the User-Defined phrase list after they have been used successfully in a word completion. In this way, the User-Defined phrase list will become filled with the words that you use most often, and the Auto-Complete feature will become more tuned to your work style.

Ignore case when matching the typed text to a trigger sequence or phrase
Use this option to indicate whether or not character case should be considered when comparing typed text to User-Defined trigger sequences and phrases.

Beep after the trigger key has been used to cycle through all matching phrases
This option controls whether or not an audible beep should occur after the Trigger Key has been used to cycle through all potential matches.

When a delimiter triggers a user-defined phrase, insert the delimiter into the text as well
This option controls how a delimiter character should be treated when it is used to trigger a User-Defined phrase.

Set Trigger Key

Use the *Set Trigger Key* button to change the key that is used to complete a partially typed word, or expand a User-Defined phrase with trigger style activation.

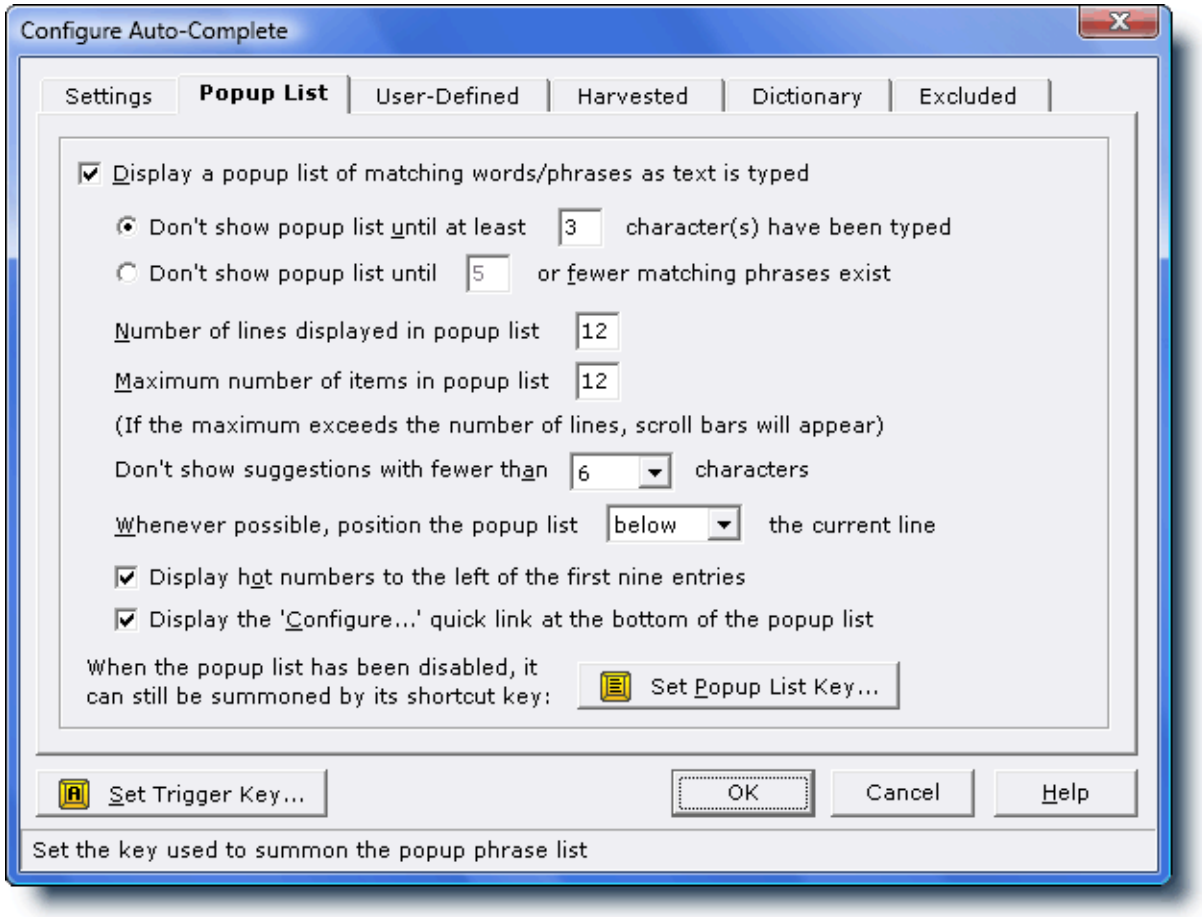
4.9.15 Auto-Complete - Popup List

Menu: Configure > Auto-Complete

Default Shortcut Key: none

Macro function: none

The Popup List tab of the Configure Auto-Complete dialog contains options that relate to the presentation of the popup list of matching words.



Display a popup list of matching words/phrases as text is typed
Use this checkbox to indicate whether or not the popup list box should appear while text is being typed. When disabled, the popup list can still be displayed using the Auto-Complete List command.

Don't show popup list until at least *n* characters have been typed
Use this option if you prefer that the display of the list be determined by the number of characters that have been typed. When this option is used, the popup list will appear as soon as '*n*' characters have been typed, provided there are matches available.

Don't show popup list until *n* or fewer matching phrases exist
Use this option if you prefer that the popup list not be shown until the number of matches falls *below* a certain threshold. When this option is used, display of the popup list will be suppressed until enough typing has occurred to narrow the list to a given number of matching words.

Number of lines displayed in popup list
This option controls the number of visible entries in the popup list -- ie, the height of the list.

Maximum number of items in popup list
This option controls the maximum number of items that will appear in the list. If this values exceed the number of lines, a vertical scroll bar will be added to the list.

Don't show suggestions with fewer than *n* characters
Use this option to specify the shortest word that should appear in the popup list.

Whenever possible, position the popup list above/below the current line
Use this option to specify where the popup list will appear in relation to the current line. When the current line is near screen top or screen bottom, the list may need to be moved to keep it on-screen.

Display hot numbers to the left of the first nine entries
This option is used to control the display and recognition of hot numbers within the popup-list.

Display the "Configure..." quick link at the bottom of the popup list
This option is used to control the display of the hot link at the bottom of the list.

Set Popup List Key
Use this button to change the key used to force display of the popup list. See the Auto-Complete List command for full details.

Set Trigger Key
Use the *Set Trigger Key* button to change the key that is used to complete a partially typed word, or expand a User-Defined phrase with trigger style activation.

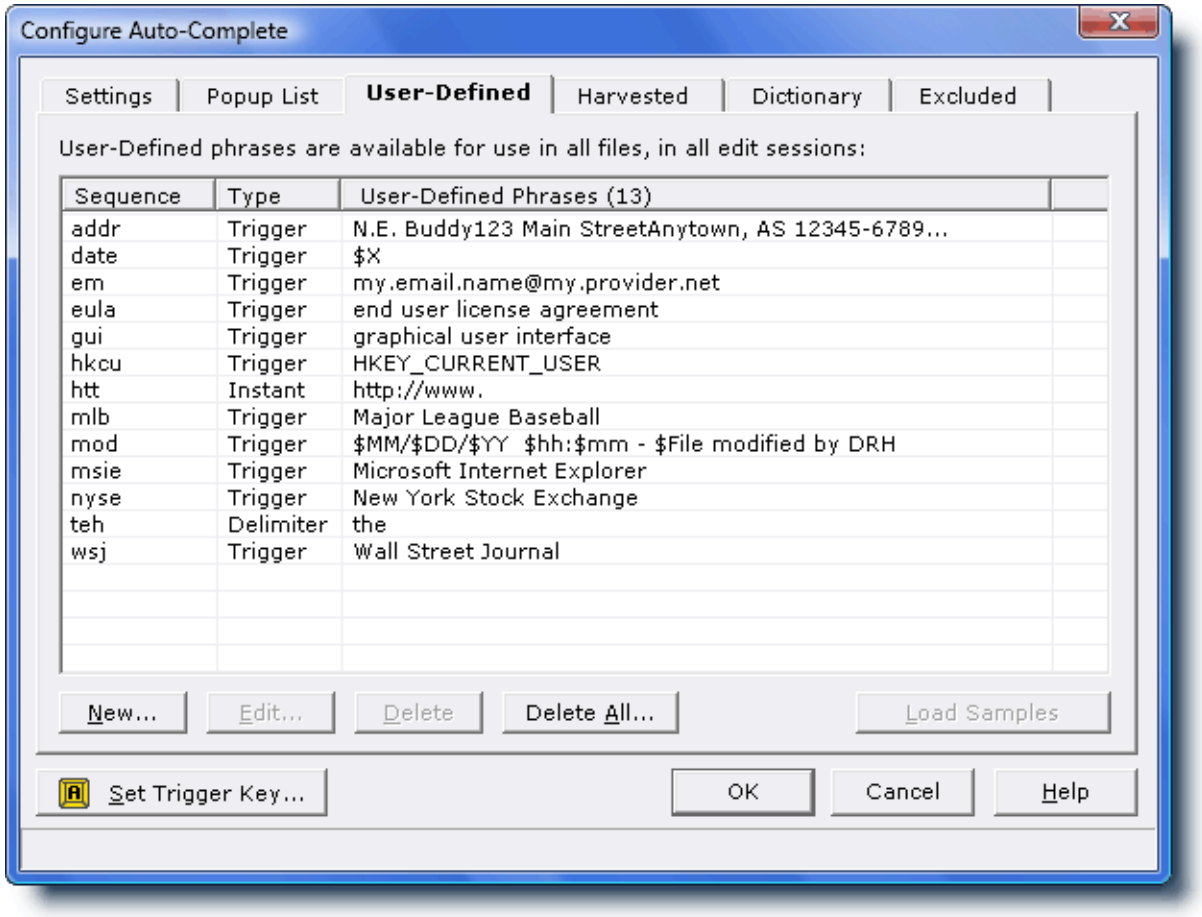
4.9.16 Auto-Complete - User-Defined

Menu: Configure > Auto-Complete

Default Shortcut Key: none

Macro function: none

The User-Defined tab of the Configure Auto-Complete dialog allows words and phrases to be defined for use with the Auto-Complete feature.



Discussion

One of the most powerful elements of Boxer's Auto-Complete function is the ability to define words and phrases which will expand automatically, after a delimiter character is typed, or when the *Trigger Key* is typed. The phrases in the dialog pictured above provide a good sampling of the types of assignments that are possible.

Three types of user-defined phrases are available:

- Instant - these phrases will be auto-inserted the instant the sequence string is typed. In the sample phrase set, typing `'htt'` will cause the string `'http://www.'` to be inserted.

- Trigger - these phrases will be inserted only when the sequence string is typed followed by the Trigger Key. In the sample phrase set, if `'eula'` is typed, and the Trigger Key is typed, `'end user license agreement'` will be inserted.

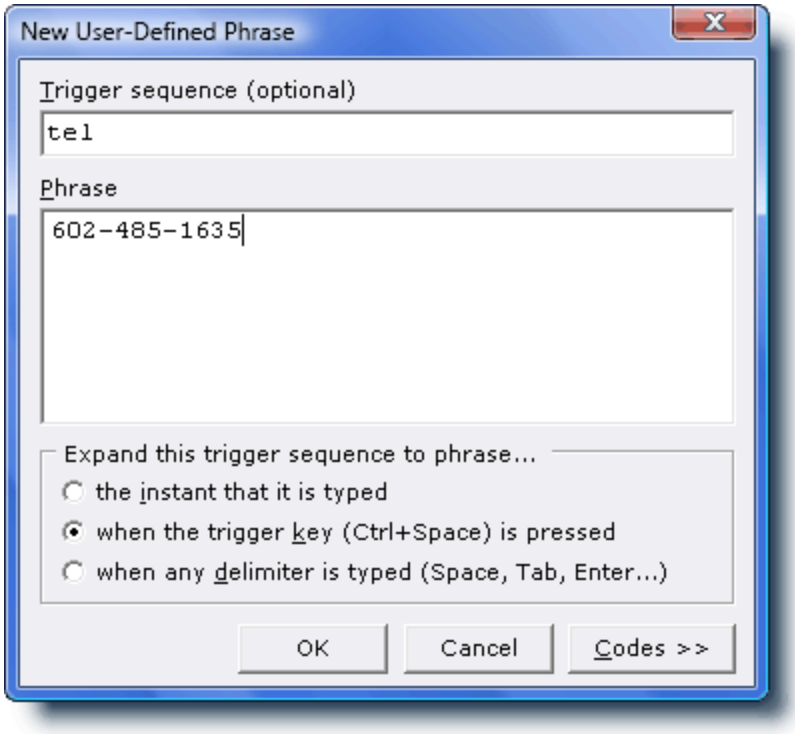
- Delimiter - these phrases will be auto-inserted only when the sequence string is followed by a delimiter character. In the sample phrase set, typing `'teh'` followed by (say) space will cause `'the'` to be inserted -- an auto-correction for mistyping `'the'`.

These three activation styles provide both utility and flexibility. Some phrases are best defined as *Instant*, while others are naturally more suitable to being *Trigger* or *Delimiter* style phrases.

You might want to invent your own conventions for defining phrases. By using an obscure lead-in or trailing character in the sequence string, virtually all phrases can be defined as *Instant*. For example, if the `addr` sequence string had been defined instead as `~addr`, its activation type could have been *Instant* since there would be almost no chance of `~addr` being typed in the course of normal work. Likewise, use `addr~` as the sequence string effectively makes ~ the new *Trigger Key*.

New

Use the New button to create a new User-Defined phrase. The following dialog will appear:

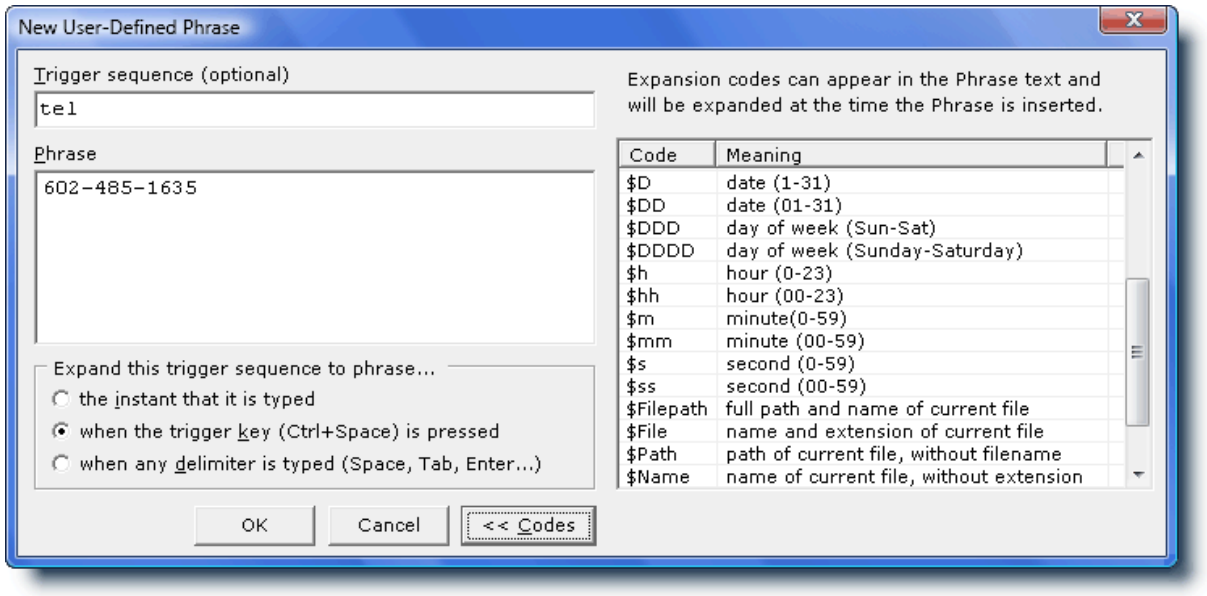


If the phrase will be triggered by a text string, enter the sequence string in the upper edit box. Enter the phrase itself in the *Phrase* memo box. Multi-line expansion phrases *are* allowed; press *Enter* to being a new line in the phrase. Finally, select the type of activation desired using the radiobuttons at the bottom of the dialog.

Expansion Codes

To define a phrase that includes *expansion codes*, click the *Codes* button to expose the

list of expansion codes:



Expansion codes will be expanded when the phrase is inserted to reflect their meaning. A variety of codes for time, date and various filename functions are available.

Edit
Use the *Edit* button to edit the settings for the selected phrase.

Delete
Use the *Delete* button to delete the selected phrase.

Delete All
Use the *Delete All* button to erase all user-defined phrases. A confirmation will be required before the operation is performed.

Load Samples
The *Load Samples* button will add a small collection of sample phrases to the list of user-defined phrases. Any phrases that are already present in the list will not be disturbed.

Set Trigger Key
Use the *Set Trigger Key* button to change the key that is used to complete a partially typed word, or expand a User-Defined phrase with trigger style activation.

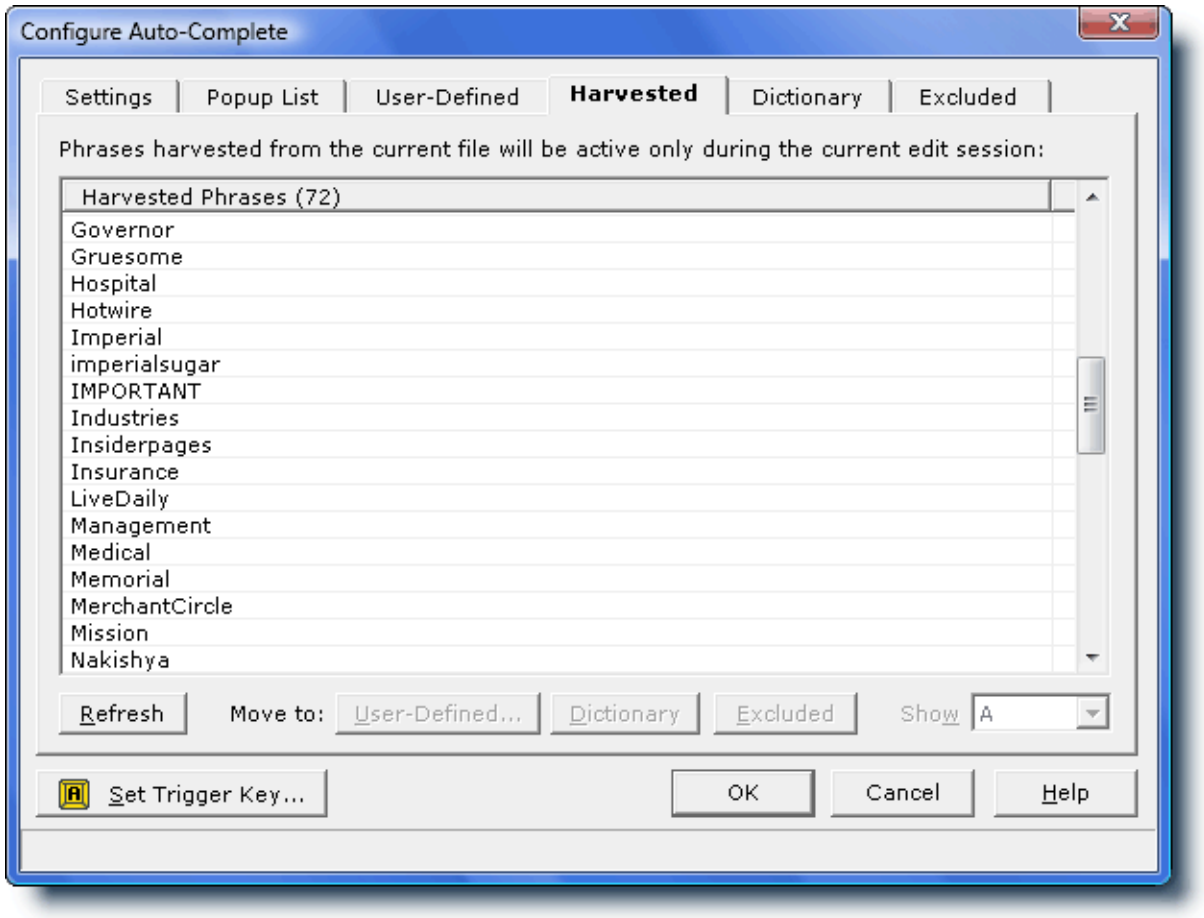
4.9.17 Auto-Complete - Harvested

Menu: Configure > Auto-Complete

Default Shortcut Key: none

Macro function: none

The Harvested tab of the Configure Auto-Complete dialog displays the words that have been harvested from the current document.



If the Harvested word list is large, the dialog will be displayed initially with an empty list. Use the *Show* option at the lower right to select the starting letter of the words you would like to view.

The Harvested word list is a temporary and transient word list. It is built on-the-fly by analyzing the current file. When an edit session ends, the Harvested word list is deleted. If the Harvested word list contains words that you would like to make permanent, there are buttons available to move words to other lists. It is not meaningful to delete a word from the Harvested word list, because it will simply reappear in the list the next time the file is next analyzed.

Refresh
Use the Refresh button to request that the current file be re-analyzed to build the harvested word list. This option is useful to check the effect of changes made on the Settings dialog tab.

Move to User-Defined
Use the *User-Defined* button to move the selected word to the User-Defined word list.

The *New User-Defined Word* dialog will appear so that additional information can be provided.

Move to Dictionary
Use the *Dictionary* button to move the selected word to the master Dictionary. This ensures that the word will subsequently appear in the popup list of matching words, even if it does not already exist in the current file.

Move to Excluded
Use the *Excluded* button to move the selected word to the Excluded word list. Words in the Excluded word list will never appear in the popup list of matching words.

Set Trigger Key
Use the *Set Trigger Key* button to change the key that is used to complete a partially typed word, or expand a User-Defined phrase with trigger style activation.

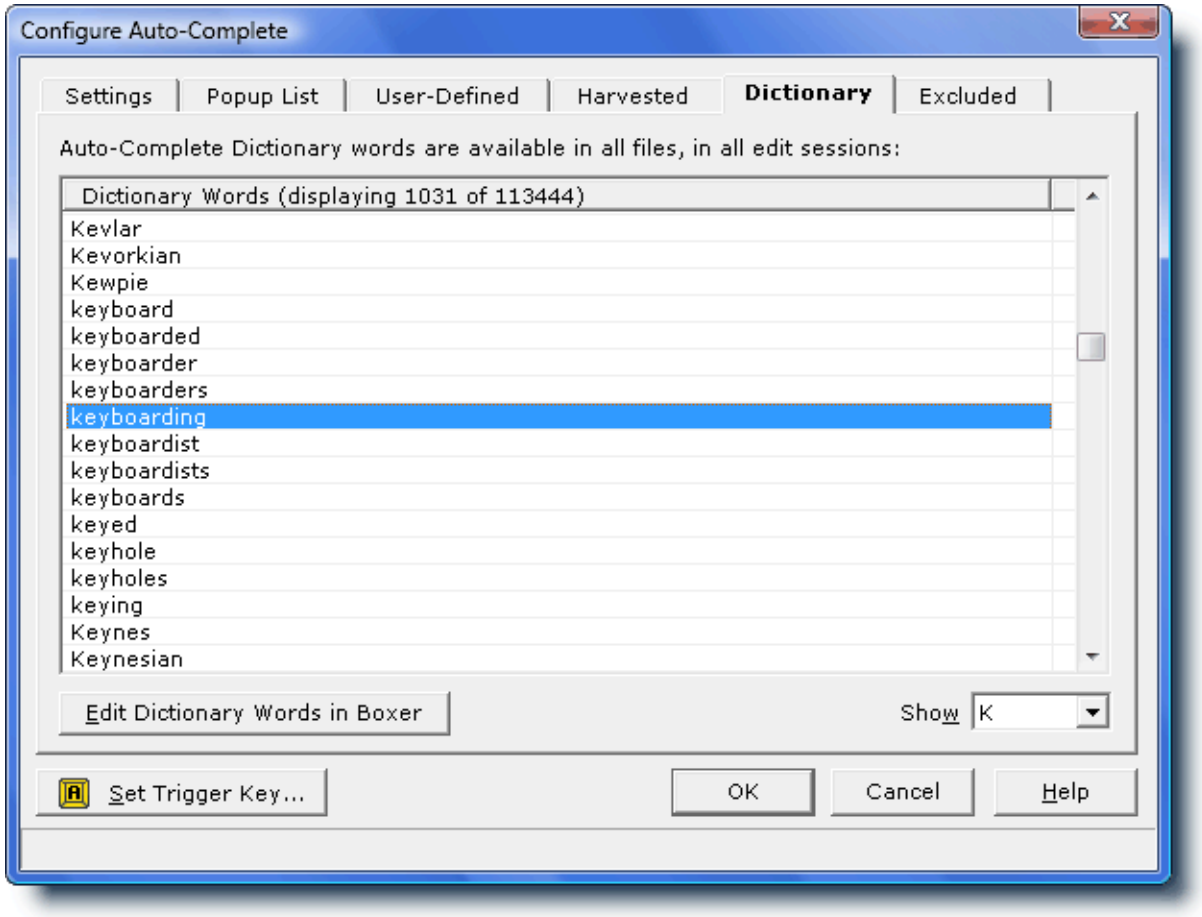
4.9.18 Auto-Complete - Dictionary

Menu: Configure > Auto-Complete

Default Shortcut Key: none

Macro function: none

The Dictionary tab of the Configure Auto-Complete dialog can be used to view or edit the Auto-Complete master word list.



Initially, the dialog will be displayed with an empty list. Use the *Show* option at the lower right to select the starting letter of the words you would like to view.

Edit Dictionary Words in Boxer
Click this button to load the dictionary word list into Boxer for viewing or editing. The word list is maintain in a simple ASCII text file, so it can be edited directly without complication. You can add new words, or remove existing words.

The format of the dictionary file is straightforward: one word per line, alphabetically sorted, case insensitive. You can use the Sort Lines command, if needed to sort the file after additions have been made.

Dictionaries for other languages are not available at this time, but if you can locate a large word list for the language of interest, you can use that list to replace the `AC_Words.txt` file provided with Boxer.

Set Trigger Key
Use the *Set Trigger Key* button to change the key that is used to complete a partially typed word, or expand a User-Defined phrase with trigger style activation.

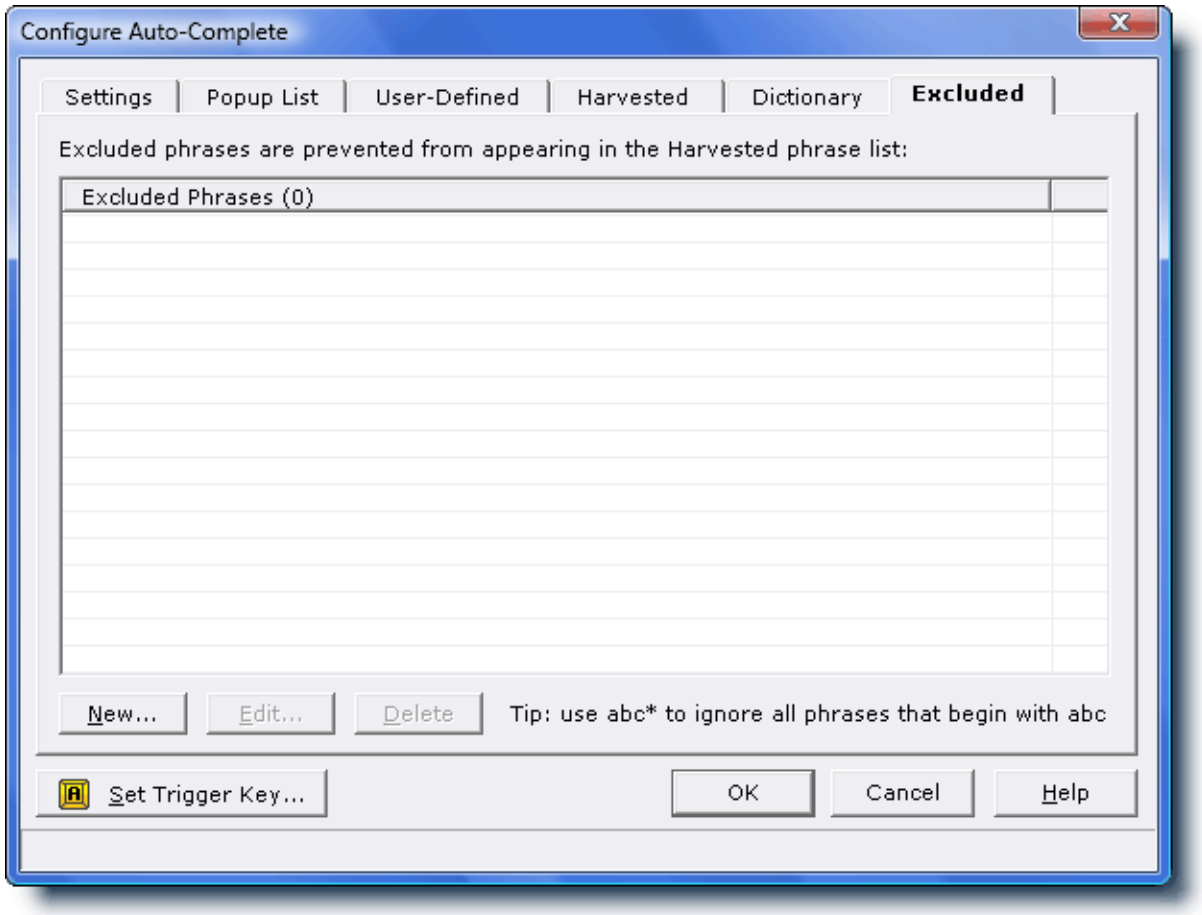
4.9.19 Auto-Complete - Excluded

Menu: Configure > Auto-Complete

Default Shortcut Key: none

Macro function: none

The Excluded tab of the Configure Auto-Complete dialog can be used to view or edit the Excluded word list. Words that appear in the Excluded word list will never appear in the popup list of matching words, even though they might otherwise be eligible to appear there. If you find that a certain word is being suggested for completion, and you find its presence in the popup list to be bothersome, simply add that word to the Excluded word list.



New
Use the *New* button to add a word to the Excluded word list.

You can use a string of the form `abc*` to cause all phrases beginning with `abc` to be excluded.

Edit
Use the *Edit* button to edit an existing word in the list.

Delete
Use the *Delete* button to delete a word from the list.

Set Trigger Key
Use the *Set Trigger Key* button to change the key that is used to complete a partially typed word, or expand a User-Defined phrase with trigger style activation.

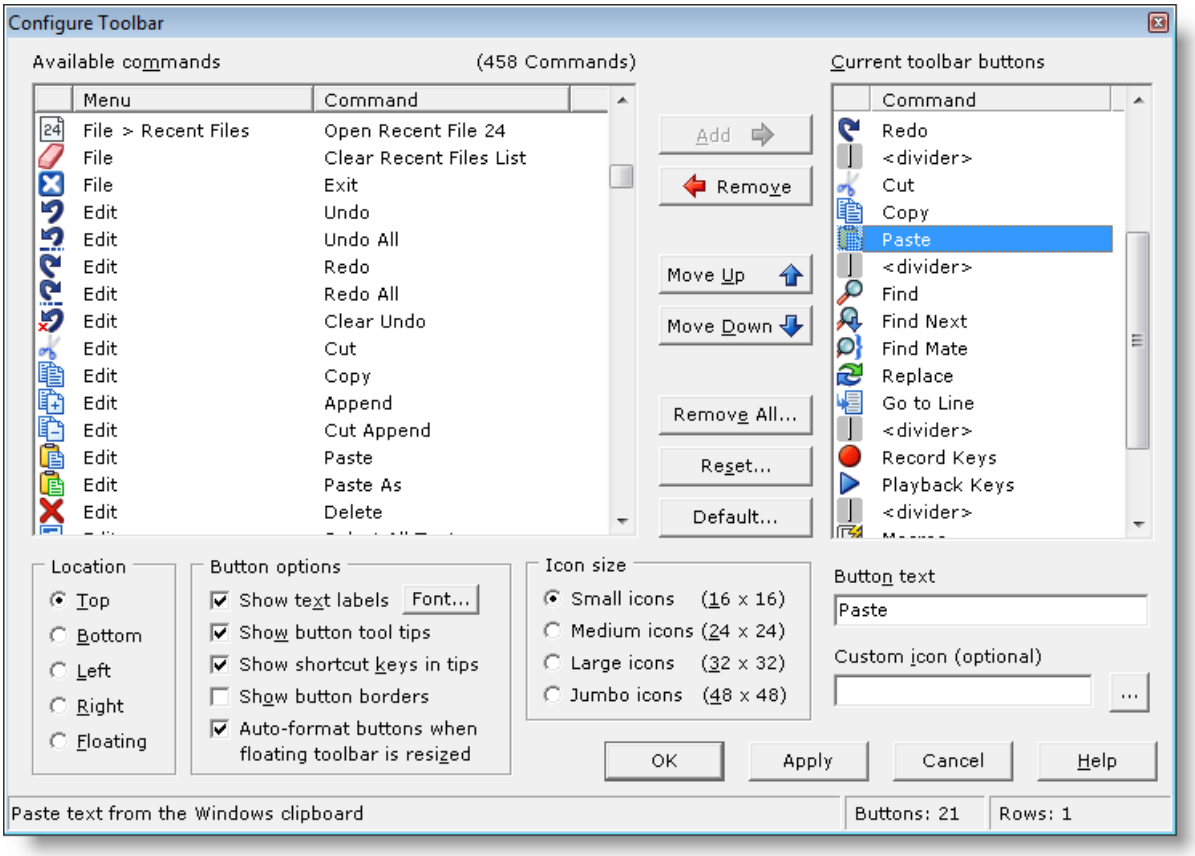
4.9.20 Toolbar

Menu: Configure > Toolbar

Default Shortcut Key: none

Macro function: ConfigureToolbar()

The Configure Toolbar command can be used to add or delete buttons from the toolbar, and to change the relative position of those buttons. Options are also available to control the location of the toolbar, whether the buttons are to have text labels, and several other features related to the toolbar.



Available commands

The *Available commands* listbox contains entries for all editor commands that are eligible for placement on the toolbar. The commands are initially presented in menu-order, but can be sorted alphabetically by clicking on the *Command* header at the top of the list. Click on the *Menu* header to return the list to menu-order presentation.

To add a new command to the toolbar, select the command in the *Available commands* listbox and click the *Add* button.

If you first select a button in the *Current toolbar buttons* listbox, you'll be able to control where a new button is added. Buttons are added below the selected button in the *Current toolbar buttons* listbox.

In addition to the editor commands, several special entries can be used to control the appearance of the toolbar. The *half space* and *full space* entries can be used to insert extra spacing between toolbar buttons. The *divider* entry can be used to insert a visible divider. The *new row* entry can be placed on the toolbar to create an additional row. All toolbar buttons situated below a *new row* entry will appear on a new row of the toolbar.

Current toolbar buttons

The *Current toolbar buttons* listbox contains entries for all of the buttons that are currently on the toolbar. The order of the list controls the order the buttons will appear on the toolbar. Use the *Move Up* and *Move Down* buttons to move a selected button within the list. To remove a button from the toolbar, select the button and click the *Remove* button. The *Reset* button can be used to restore the toolbar to its most recent configuration. The *Default* button can be used to restore the toolbar to its default configuration.

Button text
This edit box provides control over the text that will appear below a toolbar button, when that option has been selected. By default, the full name of the command will be used, but this text can be changed if desired.

Custom icon
This option allows a user-defined icon to be used in place of Boxer's default icon for a given command. Some users might wish to customize Boxer's look by using special icons, or assign more meaningful icons to extra commands that have been added to the toolbar. For example: the Run Macro ## commands all use the same toolbar icon and, unless new icons were used, would differ only in the text that is assigned to each button.

Custom icons can be supplied in either icon (.ICO) or bitmap (.BMP) format. For best results, use an image that matches the size of the icons that are in use. If the custom image does not match the icon size that's in use, the image will be scaled automatically. Boxer uses the convention that the color of the pixel in lower left corner of the image defines the background color. Pixels of this color are treated as invisible, allowing the image to blend more naturally with the background color of the button on which it is drawn.

Location

Use the *Top*, *Left*, *Bottom*, *Right* and *Floating* radio buttons to control the location of the toolbar. The location can be easily changed later on by right-clicking on the toolbar, or by dragging it from its current location.

Button options

Show text labels
Use this option to control whether or not text labels will appear below each toolbar button. The *Font* button allows the font and size to be selected from the available fonts on your system.

Show button tool tips
Use this option to control whether or not *tool tips* will be displayed when the mouse is allowed to hover atop a toolbar button.

Show shortcut keys in tips
Use this option to control whether or not a shortcut key will appear within the toolbar button tool tips.

Show button borders
Use this option to place a visible border around each toolbar button.

Auto-format buttons when floating toolbar is resized
Use this option to control the formatting of the toolbar when it is displayed in floating mode. If auto-format is selected, the toolbar will ignore any *new row* entries within the toolbar and format itself to fill the size of the toolbar. If auto-format is not selected, a floating toolbar will wrap to a new row only when a *new row* entry occurs.


As you experiment with the various toolbar button options, use the *Apply* button to preview the toolbar as it is currently configured.

Icon size

Small / Medium / Large / Jumbo
Use these options to control the size of the icons displayed on the toolbar. The natural size of Boxer's built-in icons is 16 x 16; other icon sizes are achieved by scaling these icons accordingly. You'll find that the built-in 16 x 16 icons scale smoothly to 32 x 32 and 48 x 48. The 24 x 24 icon size will show imprecision for some icons, and may be most useful when a set of custom 24 x 24 icons is being assigned.

A small collection of icons have been supplied for inspiration and experimentation. These icons are from the public domain icon set created by the Tango Desktop Project.


The *Jumbo icons* setting is very large, and may be useful to visually impaired users.


Due to internal limitations, the largest image that can be loaded from an icon (.ICO) file is 32 x 32. If you are using the *Jumbo icons* option, and you don't want your image to be scaled from 32 x 32 to 48 x 48, use a 48 x 48 bitmap (.BMP) file as the source image file.

4.9.21 Syntax Highlighting

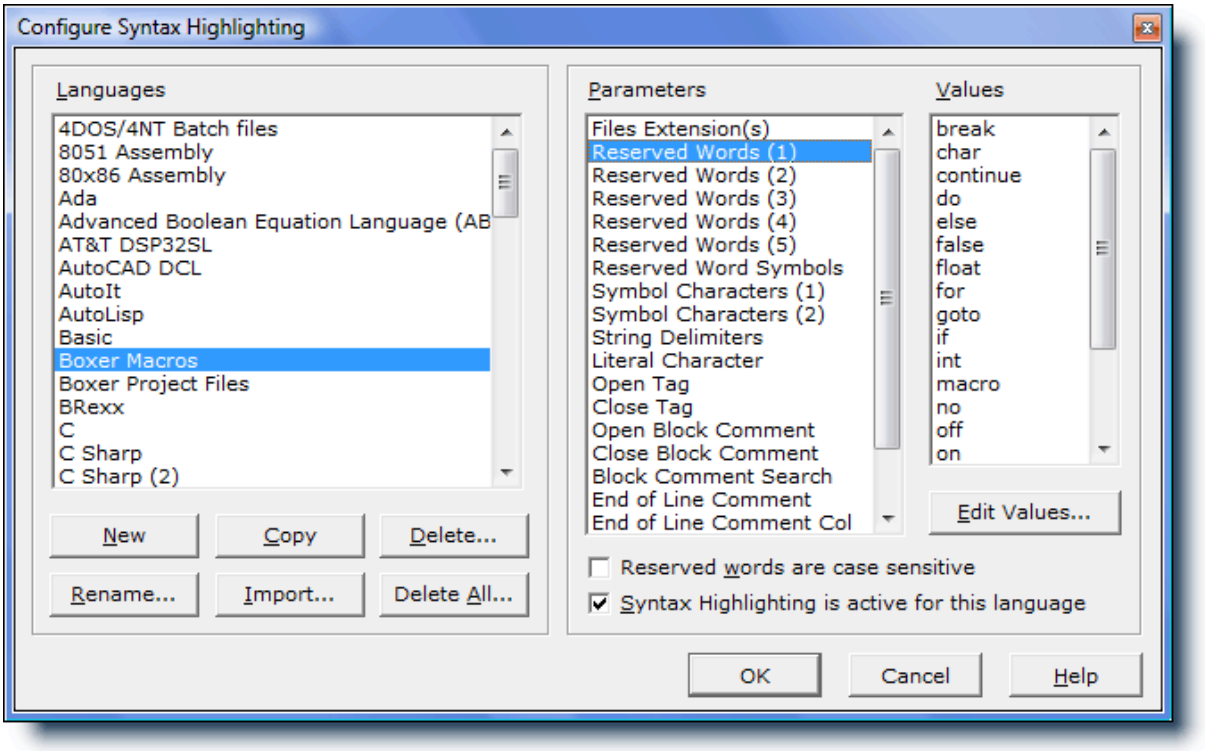
Menu: Configure > Syntax Highlighting

Default Shortcut Key: none

Macro function: ConfigureSyntaxHighlighting()

The Configure Syntax Highlighting command is used to specify the information needed by Boxer to perform on-screen Color Syntax Highlighting and Color Syntax Printing and for its other syntax-related features. Boxer is supplied with pre-defined syntax information for many popular languages. The Configure Syntax Highlighting command can be used to edit any of the pre-defined syntax information, or to define the syntax for new languages.

The dialog box below is used to define syntax information:



Boxer determines whether to perform Syntax Highlighting on the current file according to three factors.

- The extension of the current file must be one for which syntax information has been defined
- The language must be configured to be 'active' on the Configure Syntax Highlighting form (see the *Syntax Highlighting is active for this language* checkbox)
- Syntax Highlighting must be enabled on the View menu

Languages

New Button
Use the *New* button to create a Syntax Highlighting entry for a new language. An entry called *New Language* will be created which can be changed with the *Rename* button. Up to 100 different languages can be defined.

Rename Button
Use the *Rename* button to change the name of the selected language. A pop-up dialog box will appear for entering the new name. The name used will not appear anywhere other than in the list of languages.

Copy Button
Use the *Copy* button to create a copy of the currently selected language entry. The name '*Copy of...*' will be used and can be changed with the *Rename* button. The *Copy* function can save time when defining a new language which has similar characteristics to an existing language.

Import Button
The *Import* function can be used to import syntax information in the format used by our BOXER/DOS, BOXER/TKO and BOXER/OS2 products. If you have created custom syntax information with our other editors, that information can be imported directly by Boxer for Windows. First isolate the syntax information blocks which are to be imported by copying them from the `DEFAULT.CFG` file into a temporary file. Then click the *Import* button and specify that file as the name of the file to be imported. Boxer will read the named file and automatically convert the syntax information into the new format. Because the old information format did not contain a name field, you will be prompted during conversion to supply a name for each language as it is imported.

The *Import* function can also be used to import a syntax information block which has been extracted from the `Syntax.ini` file, in which Boxer stores its syntax information. This procedure may be useful for passing syntax information from PC to PC or for installing new syntax information files as they become available from Boxer Software.

Delete Button
Use the *Delete* button to delete the currently selected language. A confirmation is required before the deletion occurs. If a language is accidentally deleted, you can recover it by using the *Cancel* button.

If you simply wish to disable Syntax Highlighting for a particular language, use the *Syntax Highlighting is active for this language* checkbox described below.

Delete All Button
Use the *Delete All* button to delete **ALL** languages. A confirmation is required before the deletions will occur. You can recover from an accidental deletion by using the *Cancel* button.

USE THIS COMMAND ONLY IF YOU WISH TO DELETE ALL SYNTAX INFORMATION.

If you wish to disable syntax highlighting for all languages, use the *Perform Syntax Highlighting* option on the Configure | Preferences | Display options page. That option is non-destructive.

Parameters

The *Parameters* listbox contains all of the parameters which can be defined for a given language. Each of these parameters is discussed below:

File Extension(s)
This parameter is used to designate the file extensions which belong to the language being defined. The file extensions are named one per line, with a leading period (`.`). Be sure to include all file extensions for which highlighting is desired, such as header files, and include files. If a file type commonly goes by two names, such as `.HTM` and `.HTML`, be sure to include both extensions to guarantee that highlighting will be performed on all files desired.

 To designate that highlighting is to be applied to files without an extension, use a

lone period (`.`) on a line.

Reserved Words 1, 2, 3, 4, 5
These parameters are used to list the reserved words (sometimes known as *keywords*) which are to be highlighted. Reserved words are entered one word per line. No care need be taken to preserve an alphabetic sort, since sorting is performed automatically by Boxer.

If reserved words are to be considered case-sensitive, they should be entered in the case which is recognized by the language.

Boxer permits up to 5 sets of reserved words to be defined, and each set can be distinctly colored (see Configure | Colors). *Reserved Words 1* might be used for language keywords, such as `for`, `if`, `while`, `loop`, etc. *Reserved Words 2* might be used for preprocessor directives such as `#include`, `#define`, `#ifdef`, etc. *Reserved Words 3* might be used for library functions such as `strcpy`, `strlen`, `strcat`, etc. The *Reserved Words 4* and *Reserved Words 5* groups provide flexibility for coloring other classes of words.


The *Reserved Words 4* and *Reserved Words 5* groups do not have their own sample text entries in the miniature configuration screen in the Configure | Colors dialog. To assign colors and styles to these screen elements, select them from the *Screen elements by name* listbox.


The wildcard characters '?' and '*' are no longer recognized when defining reserved words as they were in our earlier products. We found that very few languages need this feature, while some popular languages (such as Perl) need to use '?' and '*' within their reserved words.

Reserved Word Symbols
This parameter is used to designate those symbols which are permissible within a reserved word or user variable, so that Boxer does not mistakenly highlight a phrase which happens to begin with a reserved word. An example will help clarify:

If 'read' is a reserved word, and you want to ensure that the first four letters of a variable named 'read_my_data_file' are not mistakenly highlighted as a reserved word, designate the underscore in the *Reserved Word Symbols* parameter. This tells Boxer that the underscore is allowed to appear in a reserved word or user variable, and that it is *not* a valid separator.

Alphanumeric characters are automatically permissible within reserved words. Add any additional characters which require similar treatment, one per line.

Symbol Characters 1, 2
These parameters are used to designate those characters which are to receive Symbol coloration. Two different sets of symbols are permitted, providing extra flexibility for color combinations. Designate one symbol per line.

String Delimiters
This parameter is used to designate the character(s) which are used to delimit strings.

These characters vary from language to language, but are typically the double quote and/or single quote characters. Designate one symbol per line.

☞ Boxer does not support the highlighting of strings that extend across more than one line. If you must highlight such strings, and if the language in question uses opening and closing string delimiters that are unique to one another, then you may wish to define these sequences as though they were Block Comments. Strings would then be colorized in Comment color, but multi-line strings would then be handled.

Literal Characters
This parameter is used to designate the character which is used to remove significance from an opening or closing *String Delimiter* character while within a string. Typically this is the backslash (\) character.

Open Tag
This parameter is used to designate the character which opens a tag for languages such as HTML, XML and SGML. These languages differ from conventional programming languages in that all 'code' within the file appears within markup tags, and all text outside of markup tags is considered to be the text of the document. For all other conventional programming languages, this parameter should be left blank.

Close Tag
This parameter is used to designate the character which closes a tag for languages such as HTML, XML and SGML. These languages differ from conventional programming languages in that all 'code' within the file appears within markup tags, and all text outside of markup tags is considered the text of the document. For all other conventional programming languages, this parameter should be left blank.

Open Block Comment
This parameter is used to designate the sequence (or sequences) which are used to open a multi-line block comment. Place each sequence on its own line.

Close Block Comment
This parameter is used to designate the sequence (or sequences) which are used to close a multi-line block comment. Place each sequence on its own line.

Block Comment Search
In order to properly handle multi-line comment blocks, Boxer must at times search backward in the current file to determine if a multi-line comment remains open from a line which is off-screen. This parameter designates the number of lines which should be searched during this effort. Higher values will result in better display accuracy when large block comments are used, but can slow screen display at other times.

End of Line Comment
This parameter is used to the designate the sequence (or sequences) which are used to open an end-of-line comment. An end-of-line comment persists from the point it is opened until the end-of-line. Place each *End of Line Comment* sequence on its own line.

☞ For each *End of Line Comment* defined, a corresponding *End of Line Comment*

Column must also be defined**.** See the paragraph immediately below for details.

If an *End of Line Comment* sequence includes a Space character, you'll find that comments in your text will not be not be colorized when the View Visible Spaces option is in use. This occurs because the Space character in the *End of Line Comment* sequence does not match the value of the visible space character used on screen. You can remedy this by adding a duplicate sequence that uses the visible space character in place of the Space. You can find the value of the visible space character on the Configure | Preferences | Display dialog page. This character must be entered into the edit dialog with a special technique; see the Help topic Inserting Special Character for details. Finally, remember to add the accompanying *End of Line Comment Column* parameter to mate with the duplicate *End of Line Comment* sequence.

End of Line Comment Column
This parameter is used to designate the column in which an associated *End of Line Comment* should be recognized. Some languages require that an *End of Line Comment* sequence be recognized only when it appears in a particular column, such as column 1 or column 7.

Enter the required column value, or enter 0 (zero) if the *End of Line Comment* sequence is to be recognized in all column positions. When multiple *End of Line Comment* sequences have been defined, each sequence must have a corresponding *End of Line Comment Column* entry, in the same list position as its mate.

Languages such as Clipper, dBase and FoxPro require that the asterisk (`*`) be recognized as an end of line comment when the symbol appears as the first non-blank character in the line. In other contexts the asterisk must retain its conventional meaning as the multiply symbol. This logic can be requested in Boxer (for the asterisk or any other *End of Line Comment* sequence) by using a value of -1 for the *End of Line Comment Column* parameter.

Tab Stops
Use this parameter to designate tab stop settings for files matching the *File Extensions* parameter of this language configuration. See the View | Tab Display Size command for more information about variable width tab stops.

Help File
This parameter can be used to designate an associated Windows help file (`.HLP` or `.CHM`) for the language being defined. Once the help file has been defined for a language, context-sensitive help for the word beneath the text cursor can be obtained by issuing the Help command, which is ordinarily assigned to *F1*. To obtain Boxer's native Help instead of language-specific help, simply move the text cursor into an open area of text before requesting Help. The full filepath to the reference document must be supplied.

This parameter can also be used to designate an HTML-format reference file, or indeed *any* type of reference document which the operating system knows how to open based on its file extension. For example, if you have a Microsoft Word `.DOC` file or Adobe Acrobat `.PDF` file that details the syntax of a language, these too can be named in the *Help File* parameter for that language.

The ability to display context-sensitive help for the word beneath the text cursor is available only when launching WinHelp (.HLP) and HTML Help (.CHM) files, and not when .HTML, .PDF, .DOC and other files are used.

Syntax Spell
This parameter is used to control how the Active Spell Checking feature should be applied to files which are syntax highlighted. A value of 0, 1, 2 or 3 can be used, with the effect being as follows:

0: Active Spell Checking will not be performed when editing syntax highlighted files
1: Active Spell Checking will be performed only within comments and quoted strings
2: Active Spell Checking will be performed within comments, quoted strings and 'normal' text
3: Active Spell Checking will be performed only on 'normal' text

Reserved Words are case sensitive
Use this option to designate whether the reserved word lists should be treated as case-sensitive. If this option is checked, a reserved word must match a list entry exactly in order to be highlighted. If this option is not checked, a reserved word will match a list entry even when its case is different.

This option should be selected to correspond to the requirements of the language being defined, so that Boxer can provide accurate visual feedback when a reserved word has been mistyped.

Syntax Highlighting is active for this language
Use this option to enable or disable highlighting for the current language. This option is the simplest way to disable syntax highlighting for a single language. One reason to disable a language would be to cure a file extension conflict with another language.

Use the View | Syntax Highlighting command to quickly disable syntax highlighting for *all* languages.

Notes and Tips

In addition to on-screen Syntax Highlighting, the language information defined with this command is also used for the following commands and features:

- Color Syntax Printing
- Monochrome Syntax Printing
- Syntax Spelling
- Active Spell Checking
- Auto-Complete
- Syntax Matching
- Comment
- Uncomment

If you define syntax information for new languages, or if you make additions or corrections to the pre-defined languages, please consider sending your information

to us. This will allow us to keep our information current, and make it available to other Boxer users. Syntax information can be sent to support@boxersoftware.com. Thank you in advance for your contributions.


Some users have reported using Syntax Highlighting as a teaching aid for young readers. One customer told of how she had created a syntax definition in which common nouns, verbs and adjectives were assigned to three of Boxer's reserved word classes. Then, when a file with the required file extension was displayed, each part of speech would be highlighted in its own color. Another user reported creating a 'language' definition so that headings within a dense parts list would be highlighted in color. As you can see, the uses for Syntax Highlighting extend far beyond its utility to programmers.


The highlighting of Java and Active Server code poses special problems for Boxer. These languages can include HTML markup tags as well as sections of conventional procedural style code. At times the open angle bracket (<) is a less-than symbol, at other times it could open an HTML markup tag. A rigorous handling of Java code would require that a language parser be used, which is not the method by which Boxer's (general purpose) highlighting is performed. Therefore, Boxer's default syntax information for Java has been designed to highlight Java program code, but not to highlight any HTML markup tags which might appear therein.

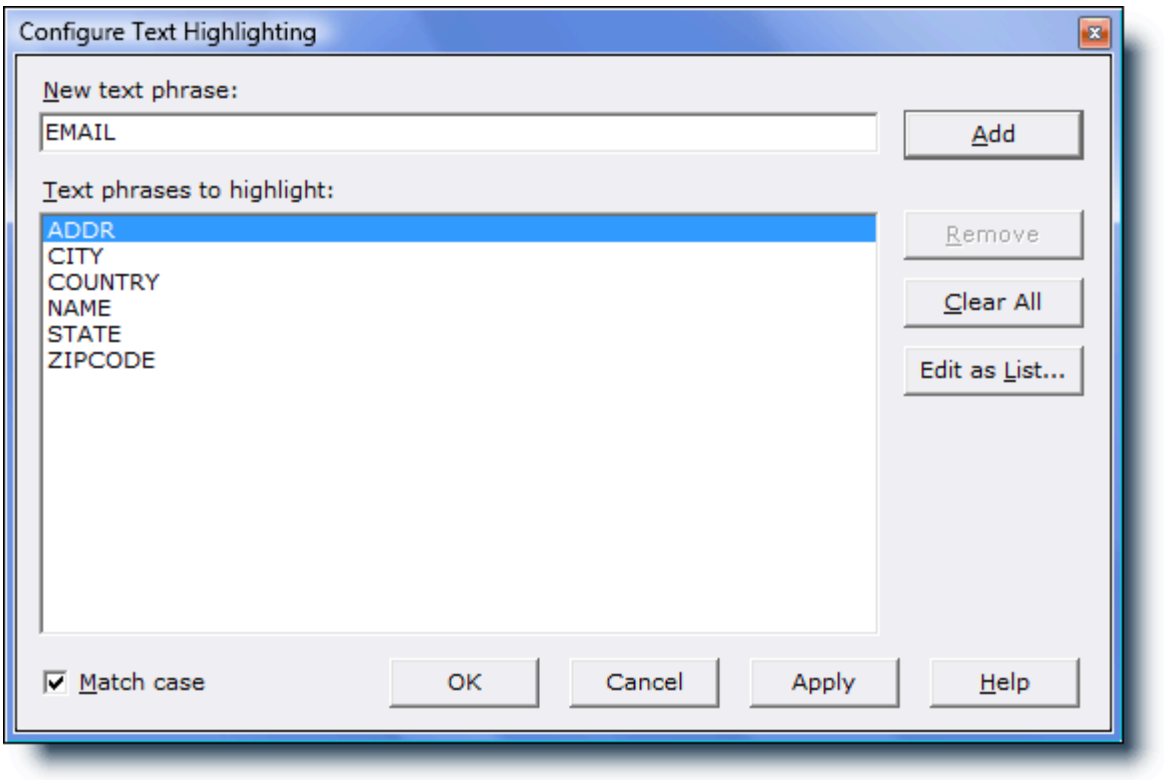
4.9.22 Text Highlighting

Menu: Configure > Text Highlighting

Default Shortcut Key: none

Macro function: ConfigureTextHighlighting()

The Configure Text Highlighting command allows the user to designate any number of text strings for on-screen highlighting. This feature might be used to make table headings stand out, or to add emphasis to any class of words or phrases that might be desired. The highlighting strings are saved and restored from session to session. The color used to highlight the designated strings is configurable on the Configure | Colors dialog. Text Highlighting can be applied to normal text files, or to program files which are already being Syntax Highlighted. The highlighting of strings can be quickly toggled on/off by using the View | Text Highlighting command.



The Find command has an option to highlight all matches of a given search string.

The Apply Highlighting command can be used to quickly add text to the list of phrases to be highlighted.

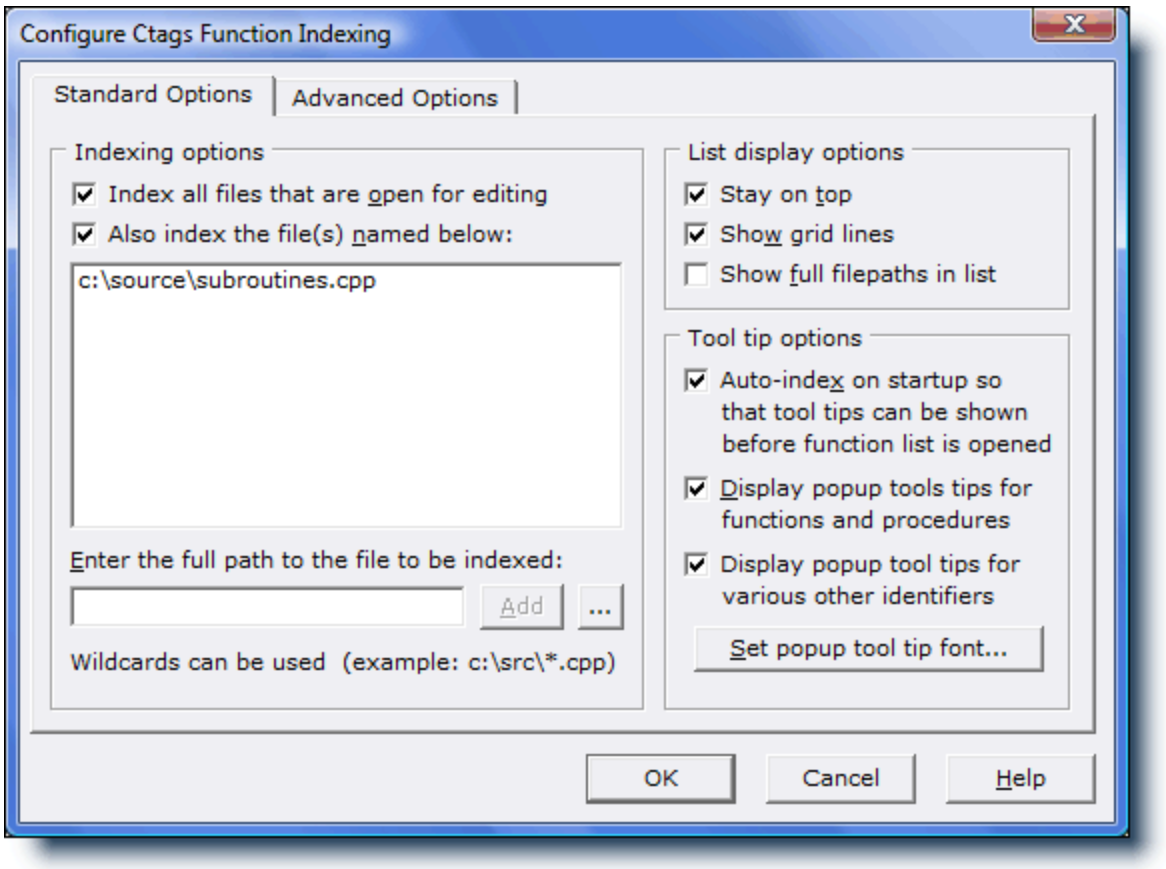
4.9.23 Ctags Function Indexing

Menu: Configure > Ctags Function Indexing

Default Shortcut Key: none

Macro function: ConfigureCtagsFunctionIndexing()

The Configure Ctags Function Indexing command provides options that relate to the Ctags Function Index feature.



Indexing options

Index all files that are open for editing
When this option is checked, indexing information will be gathered for all files that are open for editing. If the *Auto-index on startup...* option is checked (see below), this indexing will occur automatically shortly after startup, and files that are opened later in the editing session will be indexed as they are opened. If the *Auto-index* option is unchecked, indexing will not occur until and unless the Ctags Function Index command is issued.

When indexing files that are open for editing, please note that the operation is performed on the file as it resides *on disk*, and not on the memory image of the file. If you have made changes to a file that you want to be reflected in the index, be sure to save the file before requesting the indexing operation.

Also index the file(s) named below
When this option is checked, files named in the accompanying list will also be indexed, even if they are not open for editing in the editor. Use this list to name files that you would *always* like to be indexed, even when you're not editing these files. The edit box below the list is used to enter the full filepath of the file to be added. Click the *Add* button to add the file to the list. Use the button with the ellipsis (...) to browse for a file. The *Delete* key can be used to remove an unwanted entry from the list. Right-clicking in the file list will display a context menu, which additionally contains options to edit the selected entry and to delete all entries.

List display options

Stay on top
When checked, this option causes the Ctags Function Index dialog to remain on top of other windows.

Show grid lines
This option controls whether or not grid lines will be displayed between rows and columns in the Ctags Function Index.

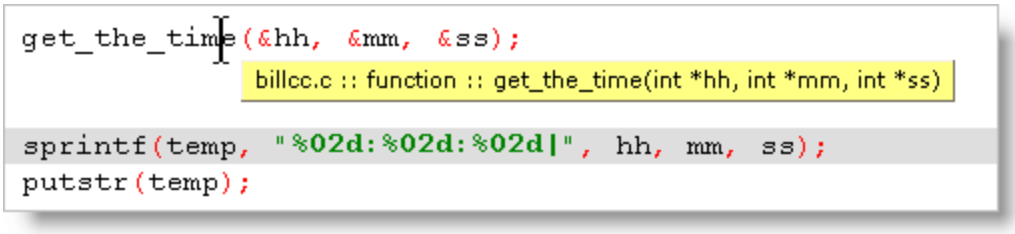
Show full filepaths in list
Use this option if you prefer that full filepaths be displayed in the Ctags Function Index. This option is useful when you're editing files that have the same filename, but reside in different directories.

Tool tip options

Auto-index on startup so that tool tips can be shown before function list is opened
Use this option to ensure that popup function prototype tool tips can be displayed even if the Ctags Function Index command has not been issued.

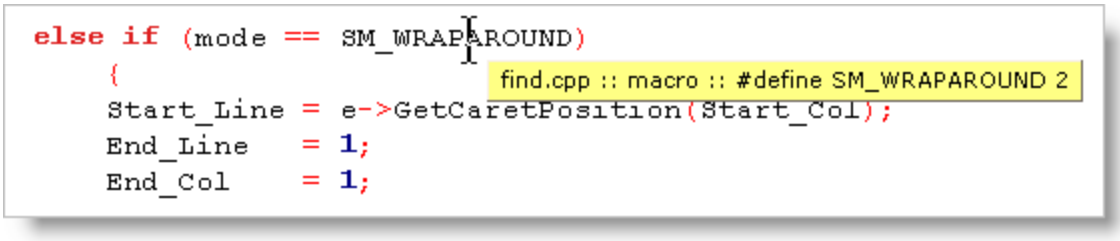

An indexing operation must be performed before function prototypes and global variable information is available for display in either the Ctags Function Index dialog, or in popup tool tips. Depending on the number of files open for editing, the number of *extra files* designated for indexing (see above), the size of these files and the processing speed of your computer, this operation could take anywhere from a split second to several seconds. On modern PC's, and with source files of modest size, the indexing process will be almost instantaneous. However, if you're using a slow PC, or you typically edit many files at once, or your source files are exceptionally large, you may wish to disable auto-indexing. For most situations, the added convenience of having popup information available will outweigh the split-second indexing process.

Display popup tool tips for function and procedures
Use this option to control whether prototypes for functions and procedures will be displayed in popup tool tips. The information that is displayed will be dependent on the language being used. For the C programming language, a popup tool tip for a function might look like this:



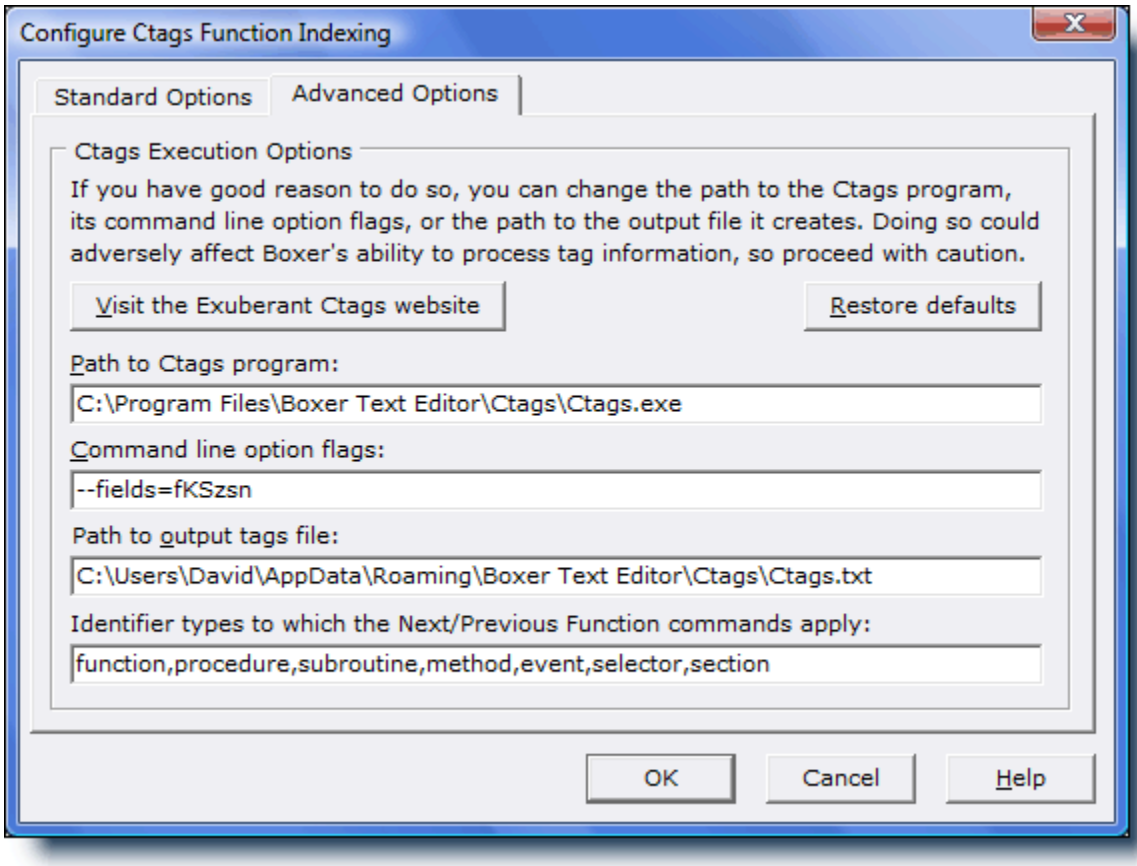
Display popup tool tips for various other identifiers
Use this option to control whether informative tool tips will be displayed for global

variables, structures, members, macros, typedefs and other recognized identifiers. The information that is displayed for an identifier will be dependent on the language being used. For example, when hovering over an identifier that has been `#defined` in the C programming language, the tool tip might look like this:



Ctags Execution Options

If you have good reason to do so, you can change the path to the Ctags program, its command line option flags, or the path to the output file it creates. Doing so could adversely affect Boxer's ability to process tag information, so proceed with caution. You can use the *Restore defaults* button to restore the settings to their recommended values.



Exuberant Ctags supports a *comprehensive* set of command line option flags. With

some experimentation, they can even be used to add support for indexing languages not supported by the program in its as-released form. Full information about the Ctags program can be found at the Exuberant Ctags website.

When run from removable media, Boxer will automatically recompute the "Path to Ctags program" and "Path to output tags file" parameters in case the drive letter of the removable device changed since the last run.

4.9.24 Templates

Menu: Configure > Templates

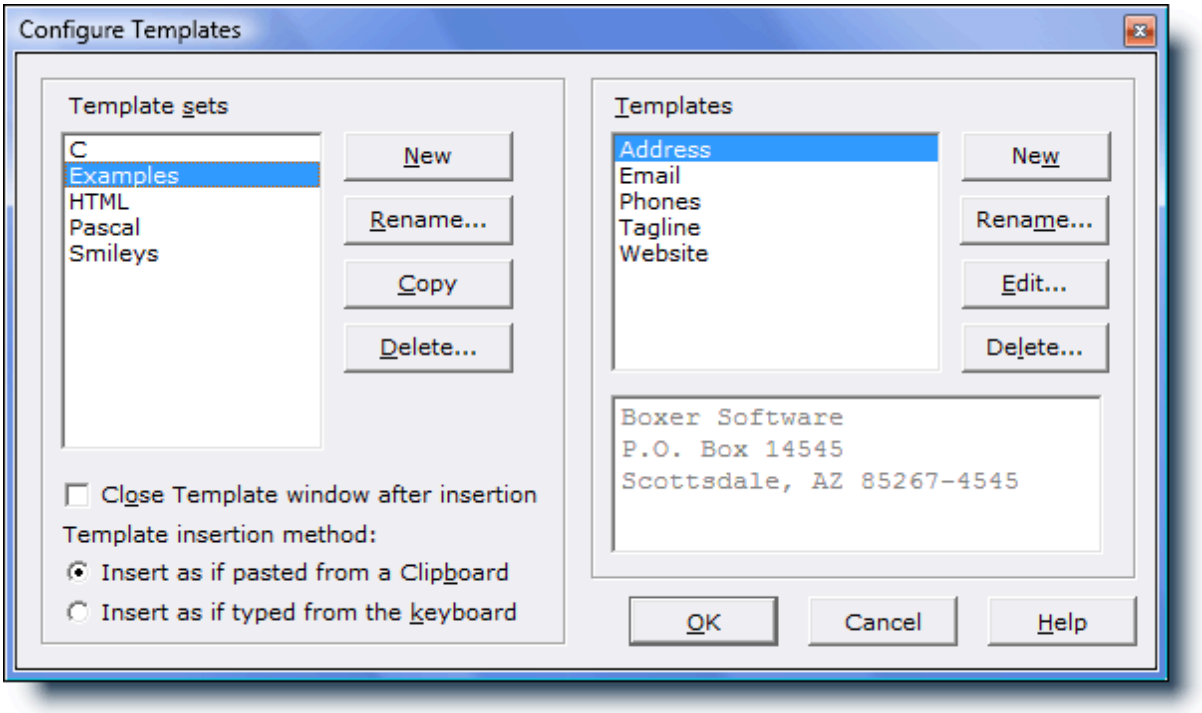
Default Shortcut Key: none

Macro function: ConfigureTemplates()

The Configure Templates command is used to create or edit *Templates*: user-defined text blocks which can be inserted into any text file from a pop-up selection menu. Once Templates have been defined, they can be inserted with the Templates command.

Templates are often used by programmers for defining the control structures of a programming language so that they can later be entered more quickly and without the chance of a typing error. The use of Templates, however, can be extended to facilitate the entry of *any* text, such as address blocks, copyright notices, phone numbers, part numbers, etc.

Template Sets and Templates are defined using the following dialog box:



Templates may consist of one or more lines of text with any formatting or indenting you choose. In addition, the template can dictate the placement of the text cursor, or how selected text should be used:

Text Cursor Placement
The Vertical Rule character (|) can be placed within a Template to dictate where the text cursor should be placed within the Template following its insertion. This allows, for example, programming code blocks to be defined in which the text cursor is placed between a pair of parentheses, ready for additional code to be typed.

Operating on Selected Text
The caret or circumflex character (^) can be placed within a Template to indicate that the template should operate on a text selection. A pair of examples will help to illustrate the power of this feature:

Example 1:
The template `<B>^</B>|` would cause the current text selection to be surrounded with HTML bold tags. The text cursor would be placed at the right of the closing bold tag.

Example 2:
The following Template:

```
<html>
<head>
|
</head>
<body>
^
</body>
</html>
```

could be run after using the Select All Text command to select the entire file. The effect would be to add the required HTML tags that help make an ordinary text file ready for viewing on the Internet. The text cursor would be place between the `<head>` and `</head>` tags, awaiting a title for the document.

Unindenting within a Template
If you need to unindent within a defined Template, use the tilde character (~) to designate the point at which the Backspace command should occur. The tilde will not be recognized in this way unless the *Insert as if typed from the keyboard* option is in force (see below).

Template Sets

A *Template Set* is a collection of Templates. Up to 100 Template Sets can be defined. Up to 500 Templates can be defined within any Template Set.

New
Use the *New* button to define a new Template Set. A pop-up dialog will appear into which the name of the Template Set is entered.

Rename
Use the *Rename* button to change the name of an existing Template Set.

Copy
Use the *Copy* button to make a copy of the currently selected Template Set.

Delete
Use the *Delete* button to delete the currently selected Template Set. A confirmation is required before the deletion occurs. If a Template Set is accidentally deleted, you can recover it by using the *Cancel* button.

Close Template window after insertion
Use this option to dictate whether or not the Template window should be closed after a Template is inserted into the edited text. Note that this option is maintained separately for each Template Set, permitting a different behavior to be defined as needed for different Template Sets.

Insert as if pasted from a Clipboard
Use this option if you prefer that Templates from the current Template Set be inserted into the text stream as if they had been pasted from a Clipboard. When this option is selected, the Autoindent setting will not influence the indent level of template text. Note that this option is maintained separately for each Template Set, permitting a different behavior to be defined as needed for different Template Sets.

Insert as if typed from the keyboard
Use this option if you prefer that Templates from the current Template Set be inserted into the text stream as if they had been typed from the keyboard. When this option is selected, the Autoindent setting will influence the indent level of template text, if the Template is inserted on an indented line. Note that this option is maintained separately for each Template Set, permitting a different behavior to be defined as needed for different Template Sets.

If you need to unindent within a defined Template, use the tilde character (~) to designate the point at which the Backspace command should occur. The tilde will not be recognized in this way unless the *Insert as if typed from the keyboard* option is in force.

Templates

New
Use the *New* button to define a new Template. First, a dialog box will be presented to get the name of the new Template. Then an editing window will appear into which the Template text can be typed.

Rename
Use the *Rename* button to change the name of an existing Template.

Edit
Use the *Edit* button to edit the content of an existing Template.

Delete
Use the *Delete* button to delete the currently selected Template. A confirmation is required before the deletion occurs. If a Template is accidentally deleted, you can recover it by using the *Cancel* button.

Boxer's Template information is stored in the file `Template.ini`, and its format is that of a simple text file, not a binary file.

If you need to insert one of the special characters (`|` or `^`) into a template in its textual form, use either `||` or `^^`. If you need to insert the special character `~` into a template, use `\~`.

If the need arises to insert a single character which is not easily typed from the keyboard, consider using the Insert Symbols feature rather than defining a single character Template. The Insert Symbols feature permits a defined character to be entered using a single keystroke.

4.9.25 User Tools

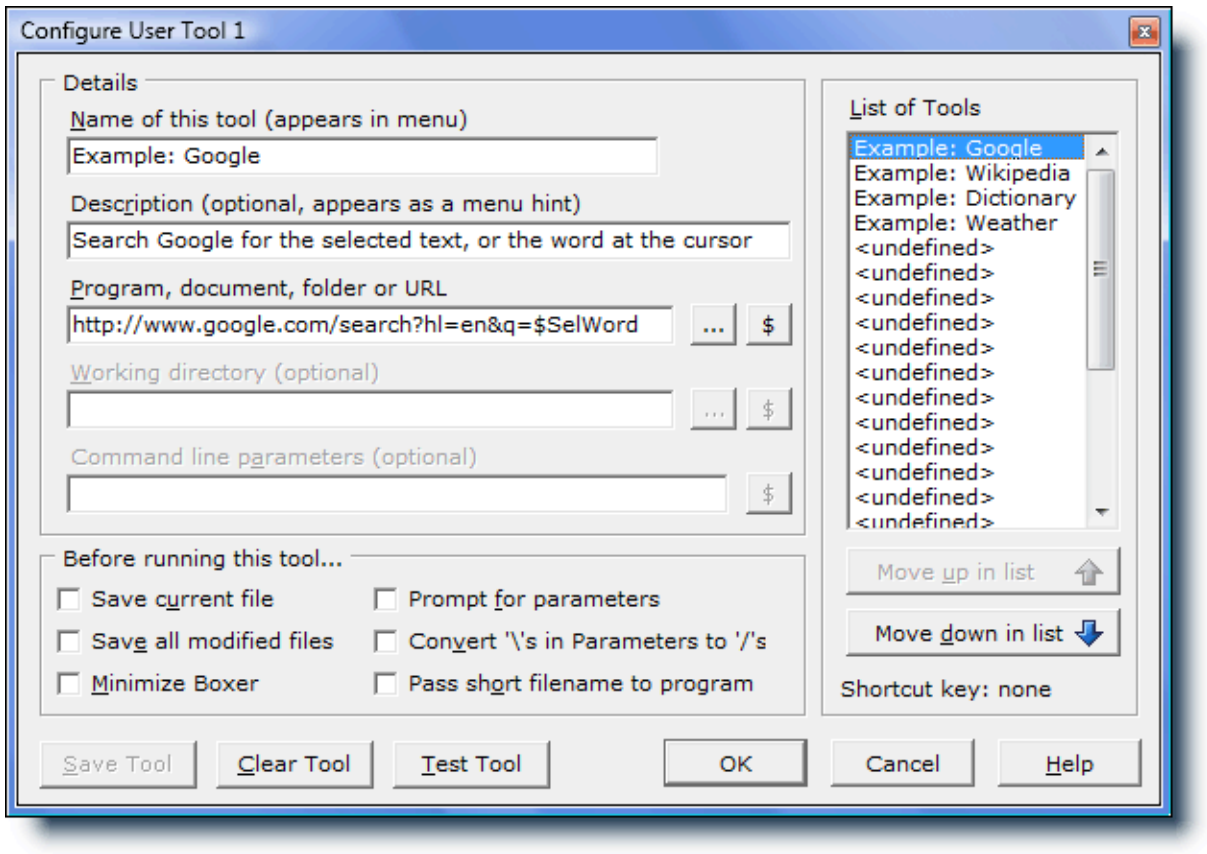
Menu: Configure > User Tools

Default Shortcut Key: none

Macro function: ConfigureUserTools()

The Configure User Tools command is used to define and configure up to 24 external programs which can be run from the User Tools submenu on the Tools menu. A variety of *Tools Macros* is available which makes it possible to control the information which is passed to the program being run.

A common use for a User Tool would be to send the name of the current file to an external program which processes the file in some way. Examples of such programs are assemblers, compilers, grammar checkers, parsers, etc.



Use of the Configure User Tools dialog box is described below:

Details

Name of this tool
Use this edit box to supply the name for the User Tool being defined. The name supplied will appear in the User Tools submenu when definition is complete. Up to 20 characters can be used.

Description
Use this edit box to supply an optional description for the tool being defined. This description will appear as a menu hint for the Tools | User Tools menu entry that corresponds to this tool.

Program, document, folder or URL
Use this edit box to supply the full filepath of the program which is to be run. The button with the ellipsis (. . .) can be used to browse for and select the desired program. If the program selected has an associated icon, it will be displayed to the right of the *Name* edit box.

Tools Macros can be placed into the *Program* field by clicking on the '$' button. For example, you might use the $SelWord directive to pass the word at the cursor--or a short text selection--to a web-based resource that performs a search for that term. The URL:

```
http://www.google.com/search?hl=en&q=$SelWord
```

would cause the word at the cursor to be sent to Google for search results.

The $Sel and $SelWord directives are processed specially when used in a URL: any embedded spaces that might result from expansion are automatically converted to plus signs (+) to create a web-friendly URL.

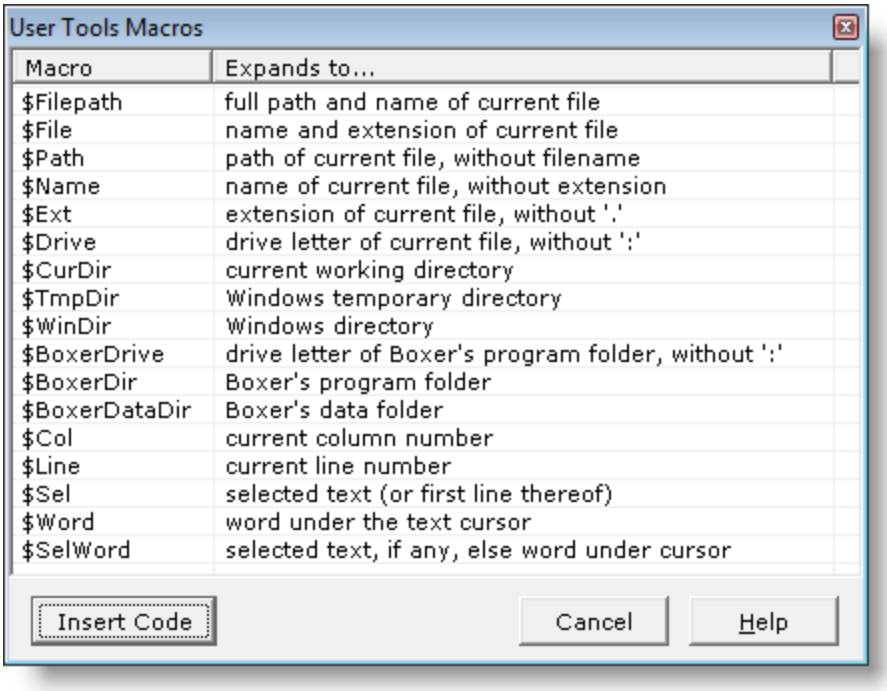
Working Directory
Use this edit box to supply the working directory for the program being defined. The working directory will become the current directory for the program being run. The button with the ellipsis (...) can be used to browse for and select the working directory.

The $Path Tools Macro (among others) can also be used in the *Working Directory* field as a means of specifying the working directory.

Command Line Parameters
The *Command Line Parameters* edit box is used to supply command line parameters to the program being defined. These parameters might be option flags required by the program or any other such information.

Boxer will recognize several different *Tools Macros* in the *Command Line Parameters* edit box to pass information to the defined program. When a macro appears in the *Command Line Parameters* field, it will be expanded to its equivalent text at the time the User Tool is run.

Clicking the ellipsis (...) button to the right of the edit box presents the *Tools Macros* list:



Macros are available to pass the current filepath, filename, extension and other portions of the filename. The line and column number can also be passed, as can the word beneath the text cursor. Selected text (or the first line of a multi-line selection) can also be passed. If needed, two or more macros can be placed in the *Command Line Parameters* edit box.

☞ If it is anticipated that the file and/or path being passed to the User Tool might contain an embedded space, be sure to enclose the $Filepath directive in double quotes to ensure proper handling.

Before running this program...

Save current file
Use this option to save any changes in the current file before running the defined User Tool. Be sure to use this option when defining a User Tool which will operate on the current file, so the program has access to the most recent changes you've made.

Save all modified files
Use this option to save any changes within all edited files before running the defined User Tool.

Minimize Boxer
Use this option to request that Boxer be minimized to the task bar while the User Tool is running.

Prompt for parameters
Use this option to request that Boxer prompt for parameters before running the defined program. This option is useful when running a program whose command line parameter(s) must be determined according to other conditions and cannot be specified

programmatically.

Convert '\'s in Parameters to '/'s
Use this option to request that any backslashes (\) within the *Command Line Parameters* edit box be converted to forward slashes (/) before running the defined program.

Pass short filename to program
Use this option to request that any *Tools Macros* used in the *Command Line Parameters* edit box be converted to *short filenames* before running the defined program. This option is needed when defining a User Tool which passes a filename to a DOS program, or to a 16-bit Windows program, since these programs are typically unable to process *long filenames*.

Buttons

Save Tool button
Use the *Save Tool* button to save the current tool definition. Note that the current tool will also be saved automatically when moving to a new tool in the *Tools* list.

Clear Tool button
Use the *Clear Tool* button to clear the definition for the current tool. No confirmation is required before the tool is erased.

Test Tool Button
Use the *Test Tool* button to simulate running the current tool, without actually executing the defined program or resource. A dialog will appear showing the various fields, after the expansion of any *Tools Macros* and other requested conversions have been made.

Move up in list / Move down in list
Use these buttons to change the order in which tools appear in the User Tools submenu on the Tools menu. Clicking on a button moves the currently selected User Tool up or down in the list. Moving a User Tool up or down in the list does not cause a change in the shortcut key assignments, if any are in use. For example: a shortcut key assigned to User Tool 2 remains assigned to the second tool in the list and does not travel with a tool which is moved through that position.

Tips and Notes

The method by which Boxer runs a User Tool program makes it possible to define User Tools which are mapped to documents, rather than programs. For example, if the 'program' to be run is defined to be an HTML document, then your Internet browser will be launched to display that file. If a .DOC file is defined as the 'program' to be run, then Microsoft Word will be launched to display the document. The browse button will present a file selection dialog box which defaults to showing .EXE and .COM files, so in order to locate documents it will need to be changed to show files of all types.

The method described in the above Tip can also be used to create User Tools which map to your favorite directories. If a directory name is defined as the 'program' to

be run, then Explorer will be launched with that directory in its view. The browse button cannot be used to select a directory name, so in order to define a User Tool in this way, the directory name will need to be typed manually into the *Program* edit box.


Some DOS programs issue an on-screen report but do not pause for user interaction or confirmation before terminating. When such programs are run as User Tools, they will execute and terminate so quickly that their results cannot be studied. You can remedy this behavior by making a change to the Properties of the program being executed. Locate the program in Explorer, right-click on its icon, and select Properties. Click the *Program* tab and uncheck the option titled *Close on exit*. Click *OK* to save the change. Thereafter, when the program is run, its window will not close until its close button is clicked.


By default, Boxer is configured to present a warning message when a file it is editing is changed by another program or process. This capability is especially useful when running User Tools since it confirms that a change was made and provides the opportunity to Reload the file from disk to get the latest copy. If you will be running a User Tool which operates on the current file, this option should be kept in force. The option is located on the Configure | Preferences | Messages option page, and is titled *Warn when an edited file is changed by another program.*


If you are a user of one of the JP Software command processors and wish to specify a `.BTM` file as a User Tool, you may need to make a system configuration change before doing so. To see if a change is required, create a `.BTM` file which performs some passive operation (such as `DIR`), and try to execute it by double clicking from within Explorer. If the file executes properly, then the Windows shell is aware that `.BTM` files can be executed, and no changes are needed. If the file does not run, then a change will be needed before a `.BTM` file can be run as a program in one of Boxer's User Tools. JP Software has documented this configuration procedure in an information file on their website. At the time of writing, this file could be found at: www.jpsoft.com/help/index.htm?deskobjs.htm If the file is not found there, look in the *Support* section at www.jpsoft.com.


Here's a tip for users of JP Software's 4DOS and/or 4NT command processors: If you would like to direct the error output from a DOS program to the Windows clipboard, you can make use of the `clip:` logical device to achieve this. At the end of any *Command Line Parameters* which might be defined for a given User Tool, add the following: `>&>! clip:` This directive causes the *standard error* stream to be placed on the Windows clipboard, overwriting the current clipboard content. The clipboard can later be reviewed in Boxer or manipulated as required.

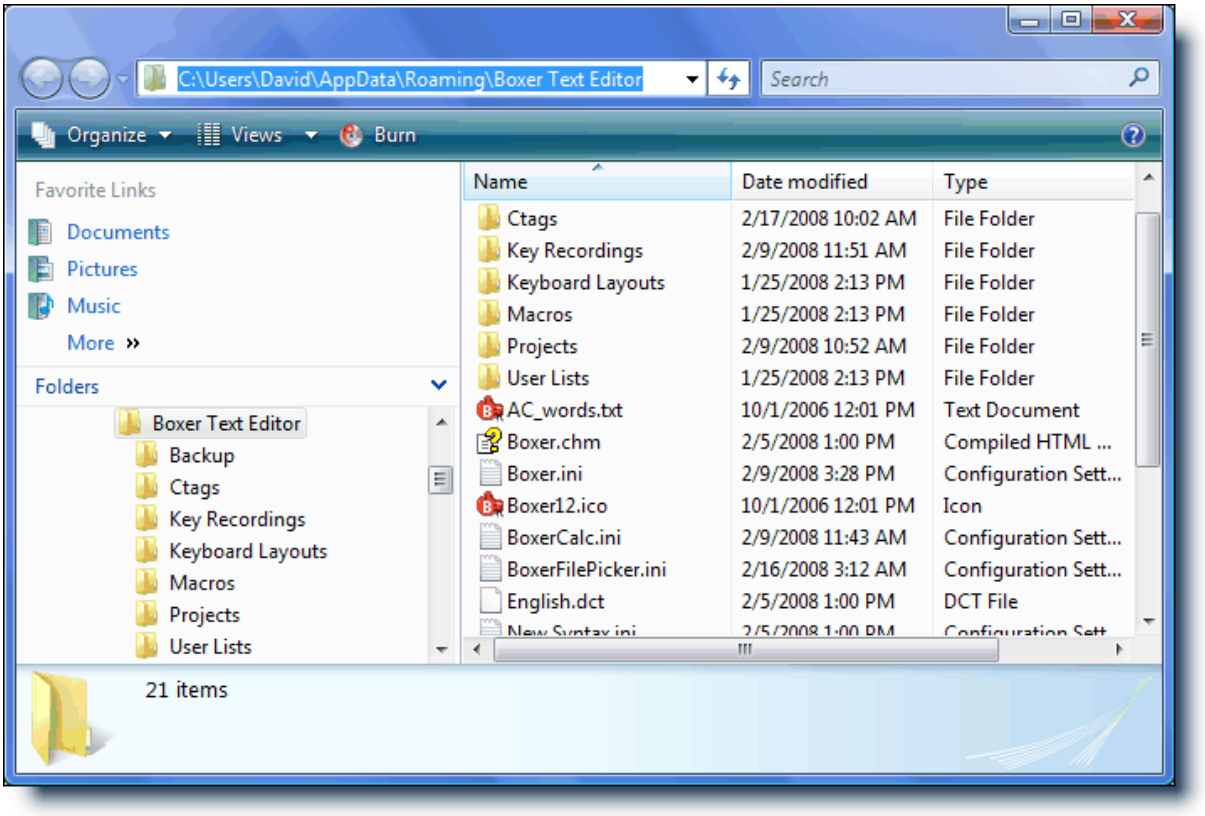
4.9.26 Explore Data Folder

Menu: Configure > Explore Data Folder

Default Shortcut Key: none

Macro function: ExploreDataFolder()

This command provides a convenient method of opening an Explorer window that points to the folder that holds Boxer's application data files. Among these are the files that contain syntax highlighting information, templates, user lists, projects, macros, as well as others.



See also the Explore Program Folder command.

When installed on an operating system prior to Windows Vista, Boxer's home/installation folder serves double duty as both its program folder and its data folder. Beginning with Windows Vista, and also under Windows 7, Boxer will maintain separate folders for its program and data files.

This command can be used to quickly locate the folder that contains the backup files Boxer creates. This folder should be emptied periodically to conserve disk space.

4.9.27 Explore Program Folder

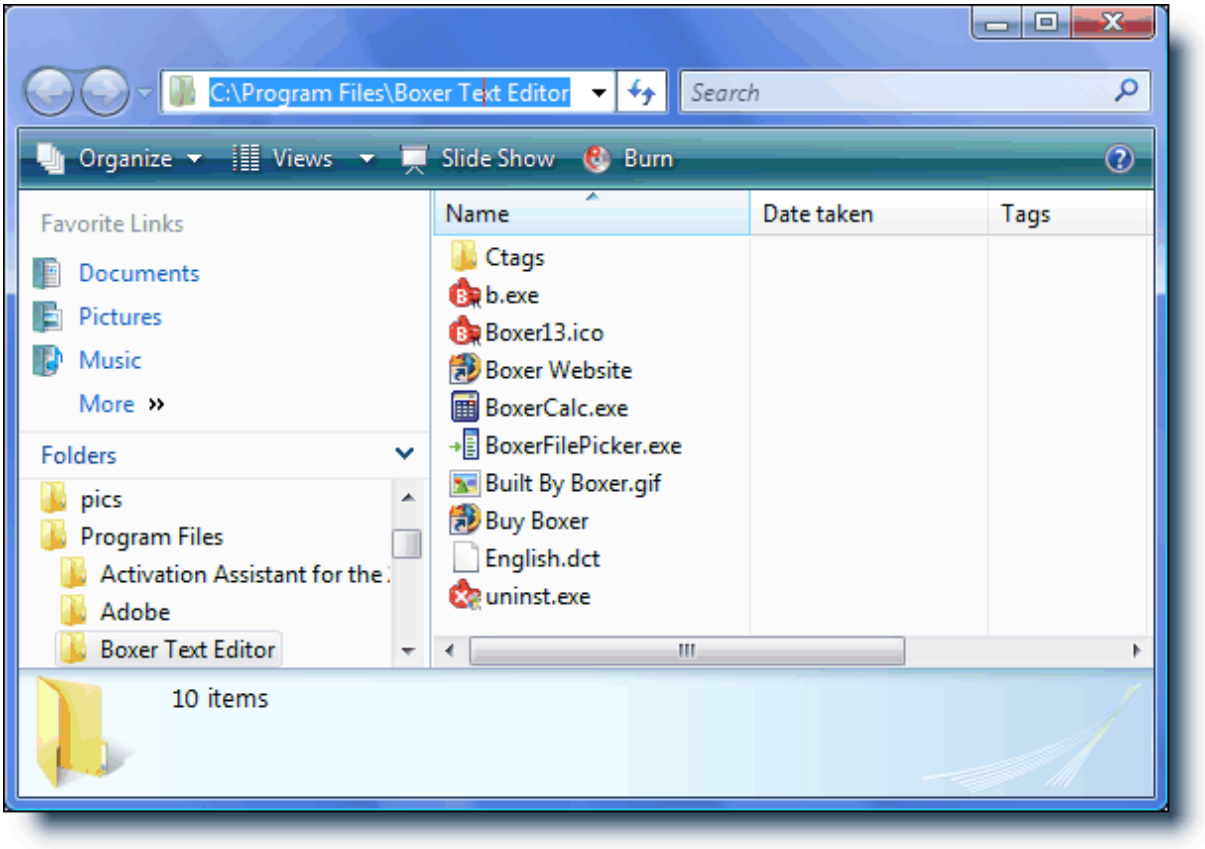
Menu: Configure > Explore Program Folder

Default Shortcut Key: none

Macro function: ExploreProgramFolder()

This command provides a convenient method of opening an Explorer window that points to the folder that holds Boxer's program files. Among these are the Boxer program

itself, calculator, uninstaller, icons and other supporting files.



See also the Explore Data Folder command.

When installed on an operating system prior to Windows Vista, Boxer's home/installation folder serves double duty as both its program folder and its data folder. Beginning with Windows Vista, and also under Windows 7, Boxer will maintain separate folders for its program and data files.

4.10 View Menu

4.10.1 Toolbar -> View Toolbar

Menu: View > Toolbar

Default Shortcut Key: none

Macro function: ViewToolbar()

The View Toolbar command is used to toggle on and off the display of Boxer's Toolbar. The Toolbar is located just below the main menu bar, and looks like this:

The Toolbar provides one-click access to Boxer's most common commands. When the mouse cursor is allowed to hover over a Toolbar button, a *Tool Tip* will popup showing the name of the command associated with that button. The display of tool tips can be disabled with an option on the Configure | Preferences | Display options page. The option is titled *Display Toolbar button tool tips*. There is also an option provided to display shortcut keys within the tool tips.

The Toolbar can be repositioned to any edge of the window by clicking on an open area in the Toolbar and dragging it to the new location. The Toolbar can also be repositioned--or turned off--by right-clicking on it to gain access to its context menu.

Toolbar buttons can be displayed in a raised, 3-D style using an option on the Configure | Preferences | Display options page. The option is titled *Display Toolbar buttons in 3-D style*.

The dollar bill icon which appears at the far right of the Toolbar is used to summon Boxer's Order Form. This icon is present only in the evaluation version of Boxer.

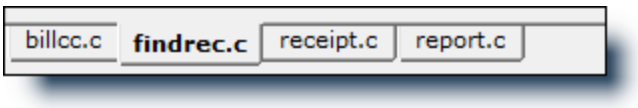
4.10.2 File Tabs -> View File Tabs

Menu: View > File Tabs > View File Tabs

Default Shortcut Key: none

Macro function: ViewFileTabs()

The View File Tabs command is used to toggle on and off the display of the *File Tabs* which can appear at the bottom or top of Boxer's window. File Tabs provide a convenient method of switching among the currently open windows.



The tab for the current file is displayed as the uppermost tab, and its name will appear in **bold** text. For example, in the picture above, the current file is `REPORT.C`. Clicking on any other tab will bring that file to the foreground position.

Right clicking on an open area of the File Tab bar will provide access to its *context menu*, which allows the bar to be repositioned or turned off. The File Tab bar can also be repositioned by dragging it to a new location.

The context menu also contains an option to sort the File Tabs alphabetically, by filename. The order of the File Tabs controls the behavior of the Window Previous and Window Next commands.

An asterisk (*) is placed in front of the filename on the File Tab to indicate that the file has changes which have not yet been saved to disk.

To reorder the file tabs, use the mouse to drag a file tab to a new location and drop it. When an edit session is resumed, the position of the file tabs will be maintained. The position of file tabs is also maintained within a project file. Note: repositioning file tabs by drag-and-drop necessitates that any file tab sorting mode (name, extension or use) which may be in force be abandoned. Otherwise, when a new file is opened and the file tabs are resorted, the drag-and-drop ordering would be lost.

The filename displayed on the File Tab can be shortened to a user-defined width. This option appears on the Configure | Preferences | Display options page.

A file can be closed by clicking its File Tab with the middle mouse button.

4.10.3 File Tabs -> Sort by Name

Menu: View > File Tabs > Sort by Name

Default Shortcut Key: none

Macro function: SortFileTabsByName()

When checked, this menu option causes the File Tabs to be arranged alphabetically by filename.

Note: repositioning file tabs by drag-and-drop necessitates that any file tab sorting mode (name, extension or use) which may be in force be abandoned. Otherwise, when a new file is opened and the file tabs are resorted, the drag-and-drop ordering would be lost.

4.10.4 File Tabs -> Sort by Extension

Menu: View > File Tabs > Sort by Extension

Default Shortcut Key: none

Macro function: SortFileTabsByExt()

When checked, this menu option causes the File Tabs to be arranged alphabetically first by file extension, and then by filename.

Note: repositioning file tabs by drag-and-drop necessitates that any file tab sorting mode (name, extension or use) which may be in force be abandoned. Otherwise, when a new file is opened and the file tabs are resorted, the drag-and-drop ordering would be lost.

4.10.5 File Tabs -> Sort by Use

Menu: View > File Tabs > Sort by Use

Default Shortcut Key: none

Macro function: SortFileTabsByUse()

When checked, this menu option causes the File Tabs to be arranged according to frequency of use. When a File Tab is clicked, the file is promoted to the first position.

Switching windows by keyboard will not cause the active file tab to be promoted to the first position. This only occurs when the file tab is clicked with the mouse.

Note: repositioning file tabs by drag-and-drop necessitates that any file tab sorting mode (name, extension or use) which may be in force be abandoned. Otherwise, when a new file is opened and the file tabs are resorted, the drag-and-drop ordering would be lost.

4.10.6 File Tabs -> Top

Menu: View > File Tabs > Top

Default Shortcut Key: none

Macro function: FileTabsTop()

Use this command to cause the file tabs to be located at the top of the screen.

The File Tab context menu also includes options to place the file tabs at screen top or bottom.

4.10.7 File Tabs -> Bottom

Menu: View > File Tabs > Bottom

Default Shortcut Key: none

Macro function: FileTabsBottom()

Use this command to cause the file tabs to be located at screen bottom.

The File Tab context menu also includes options to place the file tabs at screen top or bottom.

4.10.8 File Tabs -> Skip File

Menu: Window > Skip -*or*- View > File Tabs > Skip File

Default Shortcut Key: none

Macro function: WindowSkip()

The Skip command can be used to mark a file/window so that it will be skipped over by Window Previous and Window Next when these commands are used to cycle through open files. The skip status of each file is stored when an edit session is closed, so it will persist if the edit session is later resumed.

The File Tab context menu also includes options to toggle the skip state for the current file, or to set or clear the skip status for all open files.

Clicking on a file tab will cause that file's skip status to be released automatically, if the relevant option on the Configure | Preferences | Cursor dialog page is enabled.

4.10.9 File Tabs -> Skip All

Menu: View > File Tabs > Skip All

Default Shortcut Key: none

Macro function: none

The Skip command can be used to mark a file/window so that it will be skipped over by Window Previous and Window Next when these commands are used to cycle through open files. The Skip All command sets the skip status of *all* open files to on. You might issue the Skip All command before loading new files for editing, thereby ensuring that the Window Previous and Window Next commands would cycle only within the newly opened files.

The skip status of each file is stored when an edit session is closed, so it will persist if the edit session is later resumed.

The File Tab context menu also includes options to toggle the skip state for the current file, or to set or clear the skip status for all open files.

4.10.10 File Tabs -> Unskip All

Menu: View > File Tabs > Unskip All

Default Shortcut Key: none

Macro function: none

The Skip command can be used to mark a file/window so that it will be skipped over by Window Previous and Window Next when these commands are used to cycle through open files. The Unskip All command restores the state of and and all previously skipped windows to unskipped.

The File Tab context menu also includes options to toggle the skip state for the

current file, or to set or clear the skip status for all open files.

4.10.11 File Tabs -> Undo Close Tab

Menu: View > File Tabs > Undo Close Tab

Default Shortcut Key: none

Macro function: UndoCloseTab()

The Undo Close Tab command can be used to reopen the file that was last closed during the current editing session. This command makes it easy to reopen a file if it was closed accidentally.


The "Closed Tabs List" at the bottom of the View | File Tabs submenu shows the names of the files that are eligible to be reopened, and allows files within the list to be selectively reopened.


Clicking the middle mouse button in an open area of the file tab bar is taken as a shortcut gesture to reopen the last closed file tab.

4.10.12 File Tabs -> Undo All Closed Tabs

Menu: View > File Tabs > Undo All Closed Tabs

Default Shortcut Key: none

Macro function: UndoAllClosedTabs()

The Undo All Closed Tabs command can be used to reopen all files that have been closed during the current editing session. The names of the last ten (10) files are stored for reopening.


The "Closed Tabs List" at the bottom of the View | File Tabs submenu shows the names of the files that are eligible to be reopened, and allows files within the list to be selectively reopened.

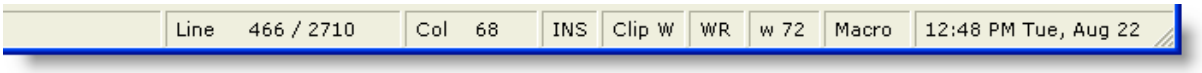
4.10.13 Status Bar

Menu: View > Status Bar

Default Shortcut Key: none

Macro function: ViewStatusBar()

The View Status Bar command is used to toggle on and off the display of Boxer's *Status Bar* which appears at screen bottom. The status bar displays the location of the text cursor in the current file, the current edit mode, current Clipboard, read-only state, Typing Wrap and Text Width settings and the current time and date.



The leftmost area of the Status Bar is used to present various information depending on the command being performed. For example, while text is being selected, a report is displayed that shows the number of lines and characters selected. The percentage of the selection, with respect to the whole file, is also displayed.

Double clicking in each of the status fields is recognized as a shorthand method of issuing a related command.

The Line Number field displays the current line number in the current file. Double clicking in the Line Number field will issue the Go to Line command. In Visual Wrap mode, the Line Number field will display paragraph numbers, since a one-to-one relationship between physical lines and screen lines no longer exists. In Visual Wrap mode, double clicking in this field will issue the Go to Paragraph command.

The Column Number field displays the current column number in the current file. Double clicking in the Column Number field will issue the Go to Column command.

The Edit Mode field indicates the current edit mode. 'INS' denotes Insert mode. 'TYP' denotes Typeover mode. Double clicking in the Edit Mode field will toggle the edit mode between Insert and Typeover modes.

The Clipboard field displays the active clipboard. 'W' indicates the Windows clipboard; internal clipboards are denoted by the digits 1-8. Double-clicking in this field advances the active clipboard by one. Shift double-clicking decreases the active clipboard by one.

To the right of the Clipboard field is the Read-Only field. If the current file is being viewed in read-only mode, 'RO' is displayed. If the file is eligible for changes, 'WR' is displayed. Double-clicking in this field will change the state of the current file within the editor. If a file is being edited in read-only mode because its on-disk read-only file attribute is set, an option is provided to change the file's on-disk read-only attribute. Changing a writable file to read-only mode does not alter a file's on-disk file attribute.

To the right of the Read-Only field is the Typing Wrap and Text Width field. Double clicking atop the 'w' in this field will toggle Typing Wrap mode on and off. A lowercase 'w' denotes off; an uppercase 'W' denotes on. Double clicking in the numeric portion of this field will issue the Text Width command.

The macro field serves several purposes. When the word 'Macro' is not flashing, double-clicking in this field will display the Macro dialog. When a macro is running, the word 'Macro' will flash intermittently. When keystrokes are being recorded using the Record Keys command, the macro field will flash the word 'Record'.

At the far right of the Status Bar is the Time and Date display. Double clicking atop the time display will issue the Insert Short Time command. Double clicking atop the date display will issue the Insert Short Date command.

Due to a problem reported by users in countries that do not use the Western/Latin *code page*, the date in the lower right corner of the status bar will now be displayed

in English, and not in the language dictated by the operating system's regional settings. The Insert Short/Long Time/Date commands will continue to honor the system's regional settings.

Unless screen space is at a premium, it is recommended that the Status Bar display be left on. Right clicking on the Status Bar summons its context menu, which allows it to be turned off.

4.10.14 Vertical Scroll Bar

Menu: View > Vertical Scroll Bar

Default Shortcut Key: none

Macro function: ViewVScrollBar()

The View Vertical Scroll Bar command is used to toggle on or off the scroll bar at the right edge on the editing window.

The height of the *thumb* or *scroll box* is proportional to the number of lines in the file. If the height of the thumb is one-third the height of the window, then the portion of the file visible within the window is approximately one-third of the entire file.

When the current file has insufficient lines to fill the height of the window, the Vertical Scroll Bar disappears automatically.

Clicking on the scroll bar with the right mouse button provides access to its context menu. The menu has an option to turn off display of the scroll bar.

As the *thumb* is dragged with the mouse, the current line and page count of the new view is displayed on the status bar in real-time. This makes it easier to locate a line/page of interest, since the current line need not be changed to get a report on the text that is in view.

4.10.15 Horizontal Scroll Bar

Menu: View > Horizontal Scroll Bar

Default Shortcut Key: none

Macro function: ViewHScrollBar()

The View Horizontal Scroll Bar command is used to toggle on or off the scroll bar at the bottom edge on the editing window.

When the current file has no lines which exceed the width of the window, the Horizontal Scroll Bar disappears automatically.

Clicking on the scroll bar with the right mouse button provides access to its context menu. The menu has an option to turn off display of the scroll bar.

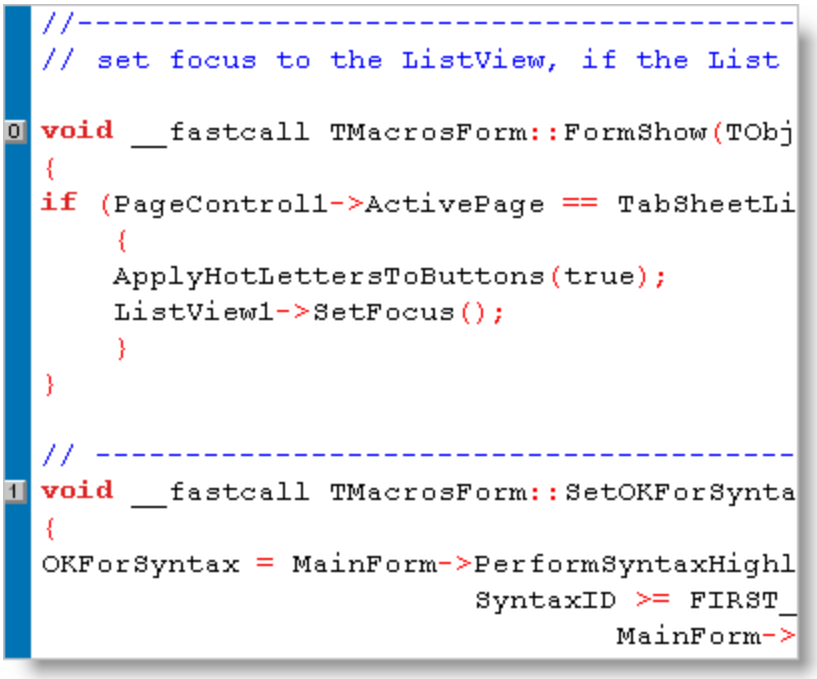
4.10.16 Bookmarks

Menu: View > Bookmarks

Default Shortcut Key: Alt+F2

Macro function: ViewBookmarks()

The View Bookmarks command is used to toggle on or off the bookmarks in the left column of the editor window. When active, bookmarked lines are displayed with a small number (0-9) in a region to the left of the editing space:



The Toggle Bookmark command is used to set or clear a bookmark on the current line at the current column of the text cursor. The Previous Bookmark and Next Bookmark commands can be used to move among bookmarked lines.

Whether or not bookmarks are displayed, they remain functional. All bookmark commands are available even when View Bookmarks is toggled off.

Clicking in the Bookmark region with the right mouse button provides access to its context menu, which allows the display of Bookmarks to be turned off.

The background color of the bookmark region is shared with that of the Line Numbers. Use the Configure Colors command to select screen colors.

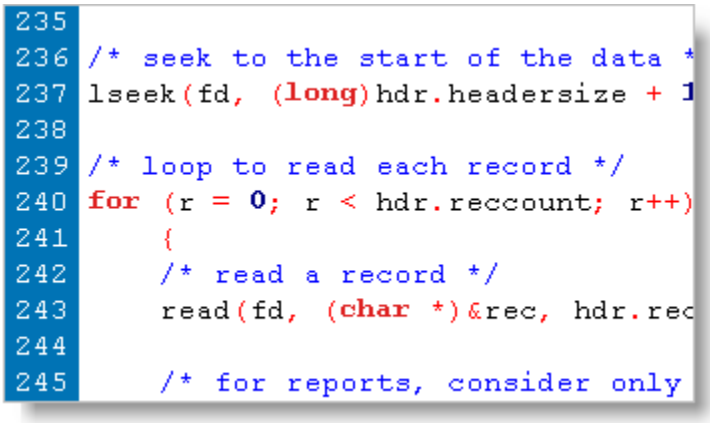
4.10.17 Line Numbers

Menu: View > Line Numbers

Default Shortcut Key: Alt+F3

Macro function: ViewLineNumbers()

The View Line Numbers command is used to toggle on or off the line numbers in a region to the left of the editing area.



In order to optimize the amount of screen space available for editing, the area allocated to the display of line numbers changes dynamically. If a file grows in size, such that the largest line number requires more space to be displayed, the line number margin will expand automatically. If a file shrinks in size, the line number margin will likewise be adjusted.

When Visual Wrap mode is active, the line number display will be adjusted to display paragraph numbers.

Leading zeros can be displayed on line numbers using an option on the Configure | Preferences | Display options page. The option is titled *Display leading zeros on line numbers*.

The display of Line Numbers is a visual aid and does not result in any changes to the file being edited. To insert line numbers into the file itself, use the Auto-Number command.

Clicking in the Line Number region with the right mouse button provides access to its context menu. Options are available to toggle on and off the display of leading zeros, and to turn off the viewing of line numbers.

The current line number is also displayed in the Status Bar.

To enable the display of a ruler which labels screen *columns*, use the View Column Ruler command.

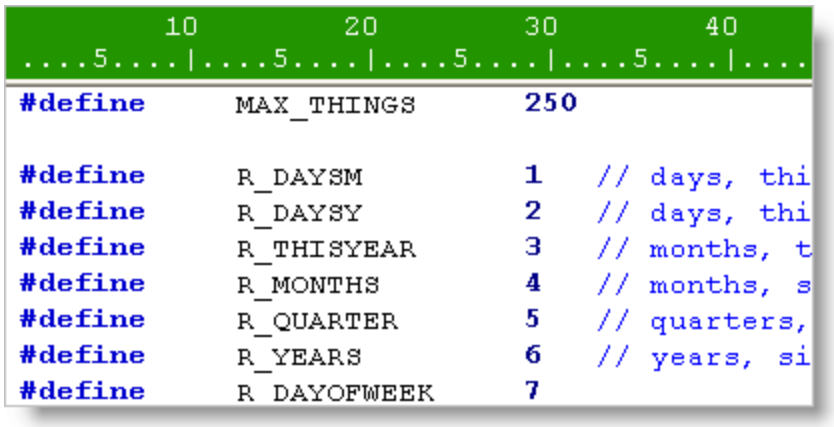
4.10.18 Text Ruler

Menu: View > Text Ruler

Default Shortcut Key: Alt+F5

Macro function: ViewTextRuler()

The View Text Ruler command is used to toggle on or off the horizontal ruler at the top of the editing window.



The Text Ruler labels the column numbers of the file being displayed. When the view of the file is scrolled to the right, the ruler values scroll along with the file. Clicking on a column number within the ruler will move the text cursor to that column on the current line. Clicking at the far right of the Ruler will cause the file to scroll to the right.

The current column number is also displayed in the Status Bar.

To enable the display of *line* numbers, use the View Line Numbers command.

Clicking on the Ruler with the right mouse button provides access to its context menu. The menu has an option to turn off display of the Ruler.

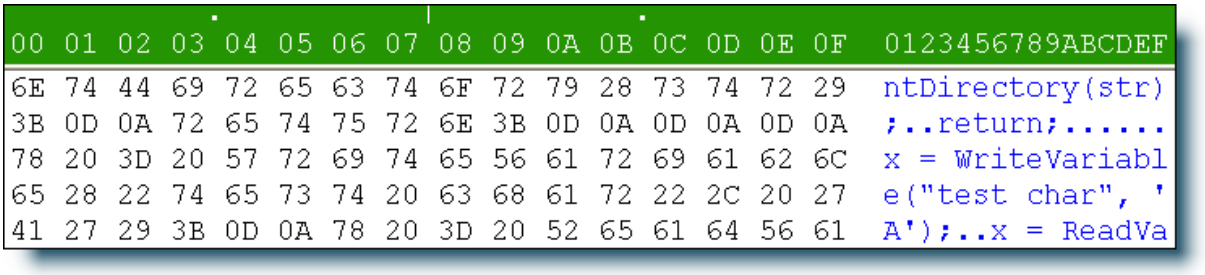
4.10.19 Hex Ruler

Menu: View > Hex Ruler

Default Shortcut Key: none

Macro function: ViewHexRuler()

The View Hex Ruler command is used to toggle on or off the horizontal ruler at the top of the editing window.



The Hex Ruler labels the column numbers of the file being displayed in hexadecimal format. Clicking on a column number within the ruler will move the text cursor to that column on the current line.

The current byte offset and column number are also displayed in the Status Bar.

To enable the display of *line* numbers, use the View Line Numbers command.

Clicking on the Hex Ruler with the right mouse button provides access to its context menu. The menu has an option to turn off display of the Hex Ruler.

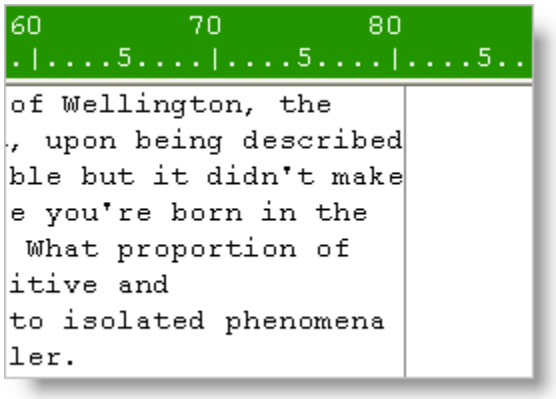
4.10.20 Right Margin Rule

Menu: View > Right Margin Rule

Default Shortcut Key: Alt+F6

Macro function: ViewRightMarginRule()

The View Right Margin Rule command is used to toggle on and off the display of a thin vertical line which marks a user-defined column. The Right Margin Rule can be used as a visual reminder that a particular line length has been exceeded.



You can set the Right Margin Rule to any column you desire; the default is column 80. An option is provided to set the column on the Configure | Preferences | Display options page. The option is titled *Show right margin rule at column...*

Clicking on the Right Margin Rule with the right mouse button provides access to its context menu, which allows the line to be turned off.

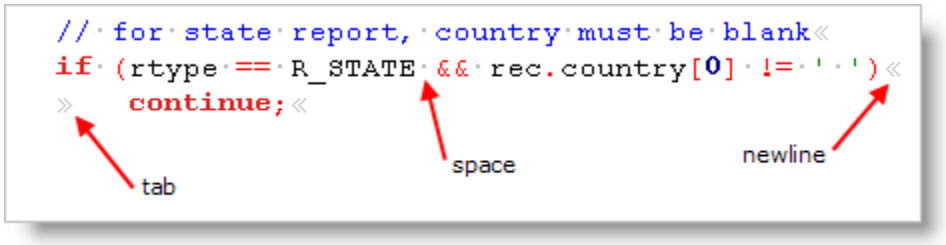
4.10.21 Visible Spaces

Menu: View > Visible Spaces

Default Shortcut Key: Alt+F1

Macro function: ViewVisibleSpaces()

The Visible Spaces command can be used to toggle on and off a display mode in which Spaces, Tabs and Newline characters (also known as *whitespace*) are displayed as visible symbols. This command is useful for drawing attention to extra Tabs and Spaces at the ends of lines, and to see whether indents are comprised of Tabs, Spaces, or both.



The color used to display Visible Spaces can be controlled with the Configure Colors command. In most of the default color schemes, a color has been used which makes the characters appear less prominent than foreground text. This often makes it possible to use Visible Spaces mode full-time, without concern for a cluttered display.

The symbols which are used to represent Spaces, Tabs and Newlines are user-configurable. These can be set using options on the Configure | Preferences | Display options page. Separate options are provided for use with both ANSI and OEM screen fonts.

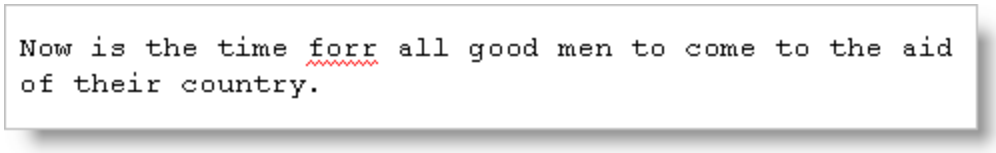
4.10.22 Active Spell Checking

Menu: View > Active Spell Checking

Default Shortcut Key: Alt+F7

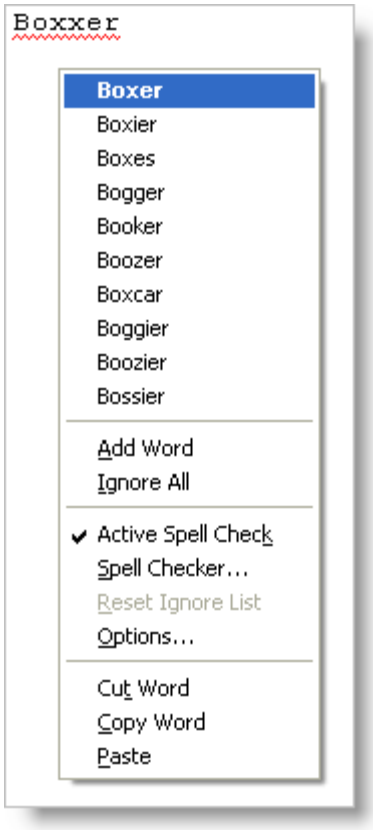
Macro function: ActiveSpellChecking()

The View Active Spell Checking command is used to toggle on/off a display mode in which misspelled words are underlined with a squiggly line. When this mode is on, Boxer will underline words as you type if they are not found in the active spell checker dictionary.



Note that Boxer does not wait for you to press *Space* and move off the word before deciding whether an underline will appear: the visual feedback is provided instantaneously. This makes it easier to experiment with alternative spellings, or to make a correction early in the word before it is completely typed.

If you right click on an underlined word, the misspelled word context menu will appear:



The context menu shows up to ten suggested corrections for the misspelled word, as well as other options related to spell checking.

Add Word
Use the *Add Word* option to add the offending word to the dictionary. Words which are added to the dictionary are saved within the file `userdict.txt` in Boxer's *data folder*. This file can also be edited within Boxer to add other words, or to remove words which may have been added mistakenly.

☞ Words which are added to the user dictionary will be accepted as correctly spelled words in any case configuration in which they may occur. For example, if the word `ebay` is added to the dictionary, it will be accepted in any of the following forms:

eBay, ebAy, and ebaY. This liberal processing was necessary because the third-party dictionary that Boxer uses is not processed in a case sensitive manner. Before this handling was put in place, the word eBay would always be reported as misspelled, even when eBay (or any variant) had been added to the user dictionary.

Ignore All
Use the *Ignore All* option to ignore the offending word, and to indicate that all other occurrences of the word should also be ignored.

Active Spell Check
Use this option to disable the Active Spell Checking feature. It can be reactivated using the View | Active Spell Checking command.

Spell Checker
Use this option to initiate a full spell checking scan with the Tools | Spell Checker command.

Reset Ignore List
This option can be used to clear the list of ignored words that have been added with the *Ignore All* option, or from earlier use of the *Ignore* button on the Spell Checker dialog.

Options
This menu entry opens the Configure | Preferences dialog to the page that contains the spell checker options.

Cut Word
Use this option to cut the misspelled word to the current clipboard.

Copy Word
Use this option to copy the misspelled word to the current clipboard.

Paste
Use this option to paste text from the current clipboard.

For general information about Boxer's spell checker, see the Spell Checker command.


The Configure | Preferences | Other dialog page contains a section with options related to Spell Checking.

4.10.23 Text Highlighting

Menu: View > Text Highlighting

Default Shortcut Key: Alt+F8

Macro function: ViewTextHighlighting()

This command is used to toggle on/off the text highlighting performed by either the

Text Highlighting command, or the *Highlight all matches* feature of the Find command.

4.10.24 Apply Highlighting

Menu: View > Apply Highlighting

Default Shortcut Key: none

Macro function: ApplyHighlighting()

This command can be used to add the word at the text cursor -- or the currently selected text -- to the list of phrases that are to be highlighted by the Text Highlighting command. This feature can be used to make table headings stand out, or to add emphasis to any class of words or phrases that might be desired. The highlighting strings are saved and restored from session to session. The color used to highlight the designated strings is configurable using the Configure | Colors dialog. Text Highlighting can be applied to normal text files, or to program files which are already being Syntax Highlighted. The highlighting of strings can be quickly toggled on/off by using the View | Text Highlighting command.


This command is also available from the context menu.

4.10.25 Syntax Highlighting

Menu: View > Syntax Highlighting

Default Shortcut Key: none

Macro function: ViewSyntaxHighlighting()

This command can be used to toggle on/off the display of Syntax Highlighting on files which are eligible for such display. This command overrides the option on the Configure | Syntax Highlighting dialog that enables and disables syntax highlighting for an individual programming language.


To create a temporary association between a file and a syntax highlighting language, or to disable syntax highlighting for a single file, use the View | Syntax Highlight As command.

4.10.26 Syntax Highlight As

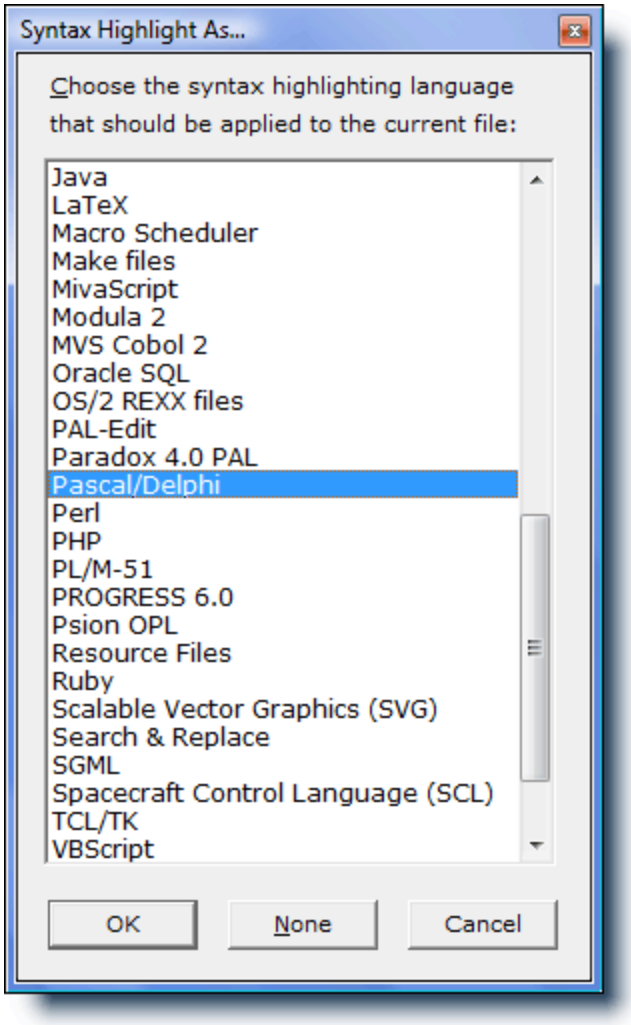
Menu: View > Syntax Highlight As

Default Shortcut Key: none

Macro function: SyntaxHighlightAs()

The Syntax Highlight As command provides a means to override the syntax highlighting that occurs due to a file's extension, or to select a language for a file that would not otherwise be eligible for highlighting. For example, if you're viewing a file named

`index.html.bak`, the Syntax Highlight As command would allow HTML to be designated as the syntax highlighting language, even though the file's `.bak` extension is not configured for HTML highlighting.



The *None* button allows a file to be disassociated from its syntax highlighting language, without the need to disable syntax highlighting for all files being edited, as the View | Syntax Highlighting command can do.

The duration of the Syntax Highlight As assignment is for the current editing session only. To permanently associate a file type with a syntax highlighting language, use the Configure | Syntax Highlighting command to add its file extension to the list of recognized extensions.

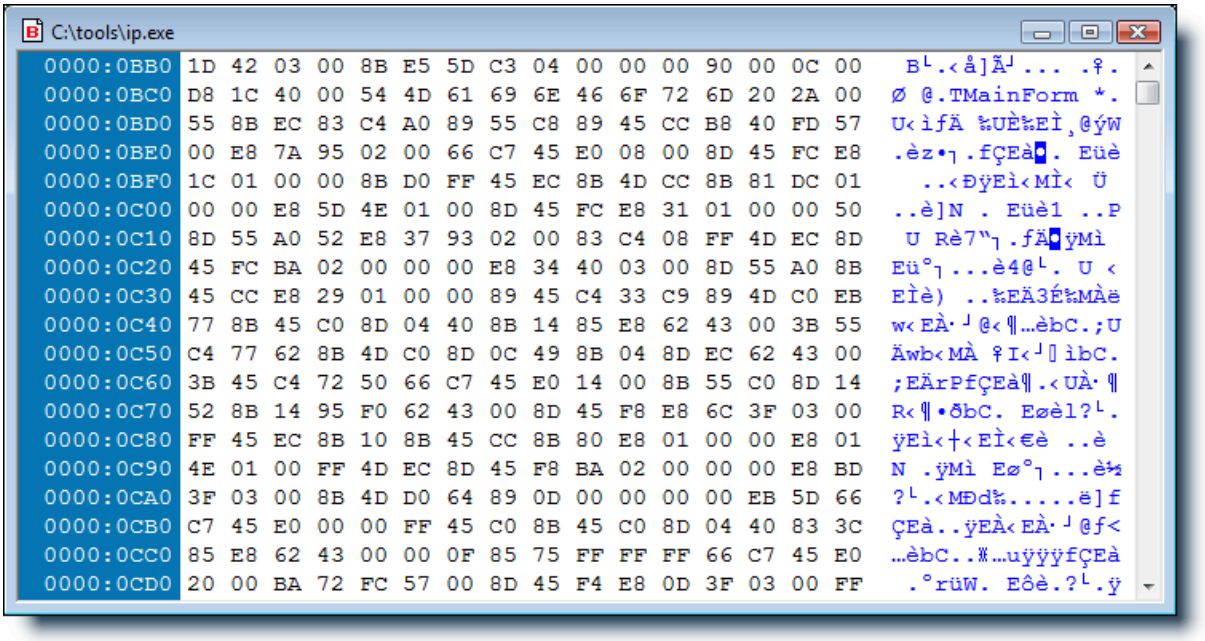
4.10.27 Hex Mode

Menu: View > Hex Mode

Default Shortcut Key: Shift+Alt+X

Macro function: ViewHexMode()

Use the View Hex Mode command to switch from normal text mode into a read-only hex mode display:



The hex mode display uses a special format which has three sections. At the left, the byte offset into the file is shown in *hexadecimal* format. In the center, sixteen bytes are displayed as two-byte hexadecimal values. At the far right the same sixteen bytes are displayed as characters, except in cases where the character cannot be so represented.

The hex mode display can be exited by issuing this command again, or by pressing *Escape*.

When switching between normal editing mode and hex mode display, the relative location of the text cursor is maintained. This makes the View Hex Mode command useful for studying the hex values characters at or near the text cursor.

The representation of the sixteen characters at the right depends upon whether an ANSI or OEM screen font is in use. The screen font can be changed with the Screen Font command.

Ordinary text files can be opened for editing and then toggled between normal and hex mode display using this command. To open a file for hex mode viewing directly--as is required for the display of binary files--use the Open Hex Mode command instead.

4.10.28 Scroll Up

Menu: View > Scroll Up

Default Shortcut Key: Ctrl+Down Arrow

Macro function: ScrollUp()

The Scroll Up command will scroll the current file up regardless of where the text cursor is positioned within the window. The cursor remains on the current line until a window edge requires it to be changed. This command is useful to scroll a file up without losing your position in the file. There must be more than a screen full of lines in order for this command to operate.

4.10.29 Scroll Down

Menu: View > Scroll Down

Default Shortcut Key: Ctrl+Up Arrow

Macro function: ScrollDown()

The Scroll Down command will scroll the current file down regardless of where the text cursor is positioned within the window. The cursor remains on the current line until a window edge requires it to be changed. This command is useful to scroll a file down without losing your position in the file. There must be more than a screen full of lines in order for this command to operate.

4.10.30 Scroll Left

Menu: View > Scroll Left

Default Shortcut Key: Alt+Left Arrow

Macro function: ScrollLeft()

The Scroll Left command will scroll the current file left regardless of where the text cursor is positioned within the window. If column 1 is already visible on screen, no movement is possible. This command is useful to scroll a file leftward without losing your position in the file.

4.10.31 Scroll Right

Menu: View > Scroll Right

Default Shortcut Key: Alt+Right Arrow

Macro function: ScrollRight()

The Scroll Right command will scroll the current file right regardless of where the text cursor is positioned within the window. This command is useful to scroll a file rightward without losing your position in the file.

4.10.32 Synchronized Scroll

Menu: View > Synchronized Scroll

Default Shortcut Key: none

Macro function: none (the interactive nature of this command makes it unsuitable for use within a macro)

The Synchronized Scroll command can be used to enter a display mode in which all open windows will scroll synchronously. This command is useful for hands-off file browsing, or for comparing similar files in side-by-side windows.

The initial direction of scrolling is downward, but the *Up* and *Down* arrow keys can be used to change direction at any time.

The *Left Arrow* and *Right Arrow* keys can be used to decrease or increase the scrolling delay, respectively.

Pressing *Right Arrow* repeatedly through the range of delay settings will set the delay to infinite. The infinite setting effectively locks all open windows to one another. The *Up Arrow* and *Down Arrow* keys can then be used to scroll all windows synchronously.

The *Home* and *End* keys can be used to move quickly to the minimum and maximum (infinite) delay settings.

Scrolling can be canceled with the *Esc* key or by pressing any key other than the arrow keys.


You may observe that Synchronized Scrolling quickens when the mouse is being moved. This is because a program receives more CPU cycles from the operating system when it is perceived to be active than when the operating system believes the program to be idle.

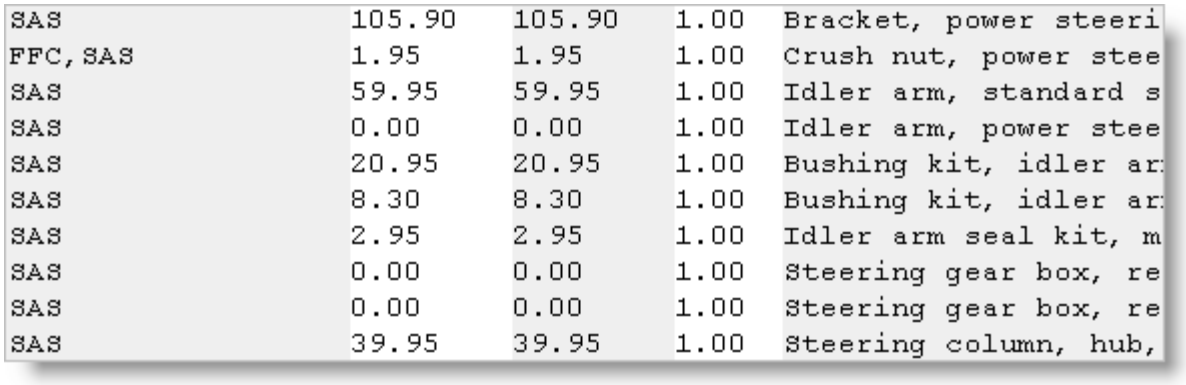
4.10.33 Shaded Tab Zones

Menu: View > Shaded Tab Zones

Default Shortcut Key: none

Macro function: ShadedTabZones()

The Shaded Tab Zones command toggles on and off a mode in which a different background screen color is used for alternating tab zones:



SAS	105.90	105.90	1.00	Bracket, power steeri
FFC,SAS	1.95	1.95	1.00	Crush nut, power stee
SAS	59.95	59.95	1.00	Idler arm, standard s
SAS	0.00	0.00	1.00	Idler arm, power stee
SAS	20.95	20.95	1.00	Bushing kit, idler ar
SAS	8.30	8.30	1.00	Bushing kit, idler ar
SAS	2.95	2.95	1.00	Idler arm seal kit, m
SAS	0.00	0.00	1.00	Steering gear box, re
SAS	0.00	0.00	1.00	Steering gear box, re
SAS	39.95	39.95	1.00	Steering column, hub,

This display mode is most helpful when Boxer is used to edit files containing character-separated field data, such as comma-separated values (CSV) or fixed-width field records. The Tab Display Size command would typically be used first to configure the proper tab stop settings for the data file. The *Intelli-Tabs* feature on that dialog is especially useful in this regard. Once the tab stop settings are entered, use this command to enable the shading of tab zones.

The alternative background color used by the Shaded Tab Zones command can be set using the Configure | Colors command. The tab zone is not depicted in the miniature screen on that dialog, but rather appears in the *Screen elements by name* listbox. Its name is *Tab Zone Background*.

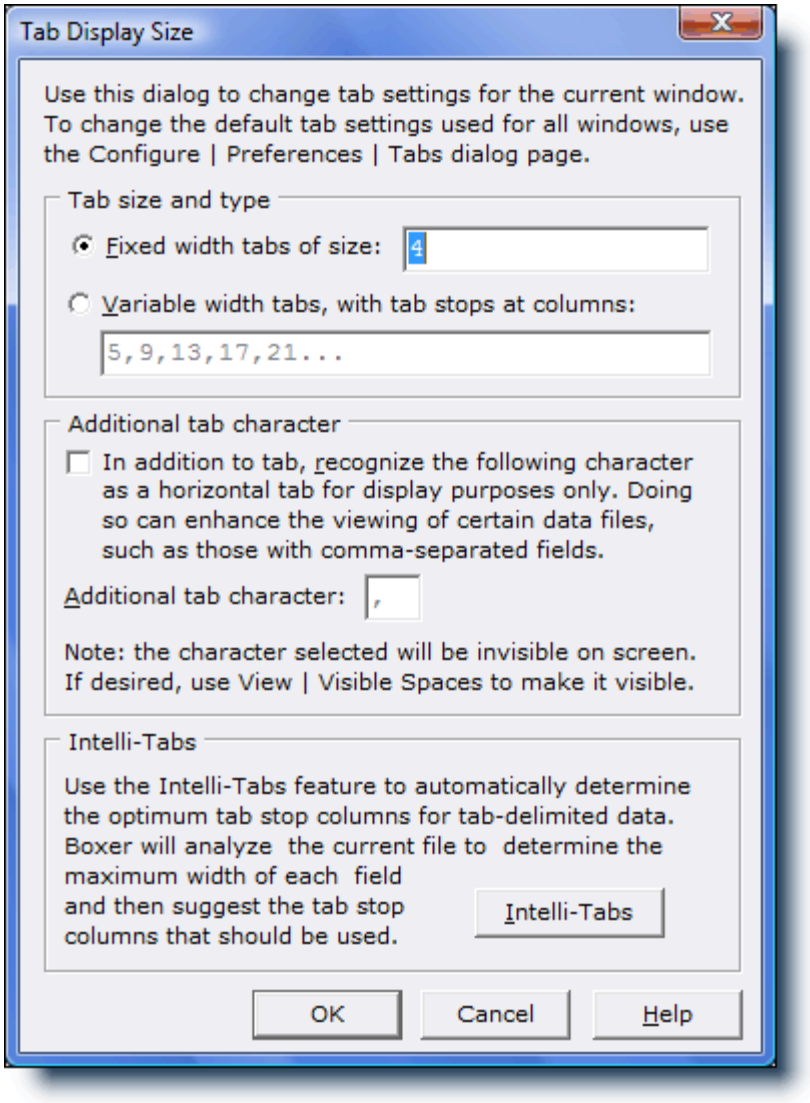
4.10.34 Tab Display Size

Menu: View > Tab Display Size

Default Shortcut Key: Alt+F9

Macro function: TabDisplaySize()

The Tab Display Size command is used to set the display width of the tab character within the current editor window. To set the default tab size for all future editing sessions, see the Configure | Preferences | Tabs dialog page.



Boxer supports the use of either fixed or variable width tabs. When fixed width tabs are used, the display width of a tab is a constant value, though the *effect* of a tab within text will depend on its column location. When variable width tabs are used, the display value of a tab is computed so as to cause a jump to the next tab stop. Variable width tabs are sometimes referred to as *typewriter style tabs*, since they mimic the function of tab stops on early typewriters.

Tab size and type

Fixed width tabs of size *n*
Use this option to set the display size for fixed width tabs in the current file.

Variable with tabs, with tab stops at columns...
Use this option to designate the columns at which variable width tab stops should occur in the current file.

Additional tab character

When viewing certain data files, it can sometimes be helpful to treat an additional character as though it were a tab, for display purposes only. For example, when viewing a file containing comma-separated fields (CSV), designating a comma as the additional tab character will allow Boxer to display the file with each field in its own column, greatly enhancing its on-screen readability. This option would typically be used when variable width tabs are in use, and most likely in conjunction with the Intelli-Tabs feature described below.

When an additional character is designated as a tab, it will become invisible on screen, just as is the tab character. You can use the Visible Spaces command to make real tabs--and the additional tab character--visible on screen.


If a comma is designated as the additional tab character, please note that it is not possible to properly process quote and comma-delimited data files whose field data contains commas within the quoted fields. Boxer requires that the field separator character appear only between fields, and not within the data itself. In the help topic for the Replace command, the *Process $1, $2, $3... substring directives in the replace string* section contains an example that shows how embedded commas can be removed.


When an additional tab character is in use, the Tabs to Spaces command will treat that character as though it was a tab. This makes it possible to convert a file that uses a character-separated field format (CSV, for example), to a fixed width field format. See the topic Converting CSV Data to Fixed Width Format.

Intelli-Tabs

Use the Intelli-Tabs feature to automatically determine the optimum tab stop columns for tab-delimited data. Boxer will analyze the current file to determine the maximum width of each data field and then suggest the tab stop columns that should be used for optimum viewing.

The Intelli-Tabs feature can also be used on files containing fixed width field data. If tabs (or *'additional tabs'*, see above) are not found in the data, fixed width field data is assumed. Then a secondary analysis is made to try to determine the boundaries of the fields. If a range of lines is selected, the secondary analysis will be restricted to the selected range.

During its analysis, the Intelli-Tabs feature will often detect records whose field count differs from those of other records. When this happens, a report will be given, and the (first) non-conforming line number will be reported. As such, Intelli-Tabs can double as a useful tool for validating data files.


Boxer's default fixed width Tab Display Size is 4, which permits program source code with several indent levels to be displayed without exceeding the screen width. Many other programs, and most printers, will treat Tabs as having a display size of 8. You may need to make adjustments in order to print or display files with another program which does not use a Tab display size of 4. One remedy could be to use the Tabs to Spaces command to convert a copy of the file before using it with the

other program. Note that Boxer's Print command will automatically convert Tabs to Spaces before sending its data to the printer, so there will be no such difficulty when printing files from within Boxer.

See the Insert Tab command for additional information about tabs.

After selecting the proper tab stops for optimum viewing, consider using the View | Shaded Tab Zones command to colorize the background of adjacent fields.

Tab settings can be designated on the command line using the -T option flag. See Command Line Options for more information.

Files that have Syntax Highlighting applied are eligible to have their tab stop settings defined as part of the syntax information for the language being highlighted. See Configure | Syntax Highlighting for more information. The parameter of interest is the *Tab Stops* parameter.

Tabs, spaces and newline characters can be made visible with the Visible Spaces command.

4.11 Window Menu

4.11.1 Tile Across

Menu: Window > Tile Across

Default Shortcut Key: none

Macro function: TileAcross()

The Tile Across command can be used to resize and reposition all editor windows such that the *client area* is fully occupied, and the windows are arranged left-to-right.

Minimized windows are not affected by this operation.

If the number of open windows will not permit a left-to-right arrangement, Windows will use an alternative arrangement which allows all windows to fit.

4.11.2 Tile Down

Menu: Window > Tile Down

Default Shortcut Key: none

Macro function: TileDown()

The Tile Down command can be used to resize and reposition all editor windows such that the *client area* is fully occupied and the windows are arranged top-to-bottom.

Minimized windows are not affected by this operation.

If the number of open windows will not permit a top-to-bottom arrangement, Windows will use an alternative arrangement which allows all windows to fit.

4.11.3 Cascade

Menu: Window > Cascade

Default Shortcut Key: none

Macro function: Cascade()

The Cascade command can be used to resize and reposition all editor windows in a cascading arrangement beginning at the upper left of the *client area* and proceeding to the lower right. The height and width of each window is uniform, and is determined by the size of the client area.

Minimized windows are not affected by this operation.

4.11.4 Cascade Vertical

Menu: Window > Cascade Vertical

Default Shortcut Key: none

Macro function: CascadeVertical()

The Cascade Vertical command can be used to resize and reposition all editor windows in a cascading arrangement beginning at the upper left of the *client area* and proceeding toward the lower left. The height and width of each window is uniform, and is determined by the size of the client area. Unlike the Cascade command, the left edges of all windows are placed against the left edge of the client area.

Minimized windows are not affected by this operation.

4.11.5 Cascade Horizontal

Menu: Window > Cascade Horizontal

Default Shortcut Key: none

Macro function: CascadeHorizontal()

The Cascade Horizontal command can be used to resize and reposition all editor windows in a cascading arrangement beginning at the upper left of the *client area* and proceeding toward the upper right. The height and width of each window is uniform, and is determined by the size of the client area. Unlike the Cascade command, the top

edges of all windows are placed against the top edge of the client area.

Minimized windows are not affected by this operation.

4.11.6 Arrange Icons

Menu: Window > Arrange Icons

Default Shortcut Key: none

Macro function: ArrangeIcons()

The Arrange Icons command can be used to neatly position minimized windows icons at the bottom of the *client area*. When several windows have been minimized, and some are later restored to normal size, the display of the remaining minimized windows can become untidy. The Arrange Icons command is useful in this situation.

4.11.7 Split Vertical

Menu: Window > Split Vertical

Default Shortcut Key: none

Macro function: SplitVertical()

The Split Vertical command can be used to split the current window vertically. A split window provides a second view into the same file, allowing two different sections of the file to be viewed simultaneously in different window *panes*. Each window pane can be scrolled separately from the other, just as if a second window were in use.

After a window is split, a thin *splitter bar* appears which visually separates the two panes. The splitter bar can be dragged left or right with the left mouse button to resize the window panes.

Clicking on the splitter bar with the right mouse button provides access to its context menu. The context menu has options to change the split from vertical to horizontal, or to turn off the vertical split so the window becomes whole again.

The Window Next command can be used to move from the left pane to the right pane, while the Window Previous command changes *focus* from the right pane to the left.

When the width of the window is increased or decreased due to window resizing, the relative position of the window split will be maintained, so long as each pane remains wider than the minimum window width.

If the Column Ruler is in use, it will appear in both the left and right window panes of a vertically split window.

☞ When the panes of a split window are resized with the mouse, a report appears on

the status line that shows the percentage of width/height allocated to each pane.

4.11.8 Split Horizontal

Menu: Window > Split Horizontal

Default Shortcut Key: none

Macro function: SplitHorizontal()

The Split Horizontal command can be used to split the current window horizontally. A split window provides a second view into the same file, allowing two different sections of the file to be viewed simultaneously in different window *panes*. Each window pane can be scrolled separately from the other, just as if a second window were in use.

After a window is split, a thin *splitter bar* appears which visually separates the two panes. The splitter bar can be dragged up or down with the left mouse button to resize the window panes.

Clicking on the splitter bar with the right mouse button provides access to its context menu. The context menu has options to change the split from horizontal to vertical, or to turn off the horizontal split so the window becomes whole again.

The Window Next command can be used to move from the top pane to the bottom pane, while the Window Previous command changes *focus* from the bottom pane to the top.

When the height of the window is increased or decreased due to window resizing, the relative position of the window split will be maintained, so long as each pane remains taller than the minimum window height.

If the Column Ruler is in use, it will appear only in the top window pane of a horizontally split window.


When the panes of a split window are resized with the mouse, a report appears on the status line that shows the percentage of width/height allocated to each pane.

4.11.9 Next

Menu: Window > Next

Default Shortcut Key: F6

Macro function: WindowNext()

The Window Next command is used to move to the next window when editing two or more files. When a window has been split with the Split Horizontal or Split Vertical commands, Window Next will move to the lower or right window pane, respectively. Window Next will skip over a minimized window, but the Windows-level service (*Ctrl+F6*) will stop on minimized windows.

The next window in the sequence will be determined according to the order of the File Tabs. If the File Tabs have been configured to sort the files alphabetically, Window Next will move to the window whose filename occurs next, alphabetically. If the File Tabs are not sorted, the order is determined
by Windows according to the "z order" ranking of the windows.

When many windows are open, it may prove faster to use the Window List to select a new window. Boxer's File Tabs can also be used to move quickly among windows.

4.11.10 Previous

Menu: Window > Previous

Default Shortcut Key: Shift+F6

Macro function: WindowPrevious()

The Window Previous command is used to move to the previous window when editing two or more files. When a window has been split with the Split Horizontal or Split Vertical commands, Window Next will move to the upper or left window pane, respectively. Window Previous will skip over a minimized window, but the Windows-level service (*Ctrl+F6*) will stop on minimized windows.

The previous window in the sequence will be determined according to the order of the File Tabs. If the File Tabs have been configured to sort the files alphabetically, Window Next will move to the window whose filename is previous, alphabetically. If the File Tabs are not sorted, the order is determined
by Windows according to the "z order" ranking of the windows.

When many windows are open, it may prove faster to use the Window List to select a new window. Boxer's File Tabs can also be used to move quickly among windows.

4.11.11 Skip

Menu: Window > Skip -*or*- View > File Tabs > Skip File

Default Shortcut Key: none

Macro function: WindowSkip()

The Skip command can be used to mark a file/window so that it will be skipped over by Window Previous and Window Next when these commands are used to cycle through open files. The skip status of each file is stored when an edit session is closed, so it will persist if the edit session is later resumed.


The File Tab context menu also includes options to toggle the skip state for the current file, or to set or clear the skip status for all open files.

Clicking on a file tab will cause that file's skip status to be released automatically, if the relevant option on the Configure | Preferences | Cursor dialog page is enabled.

4.11.12 Last Visited

Menu: Window > Last Visited

Default Shortcut Key: Shift+Alt+F6

Macro function: WindowLastVisited()

The Window Last Visited command provides a means to return quickly to the last window that was active before the current window was activated. When a large number of files is being edited, using the Window Previous or Window Next command may not be practical for this purpose. Issuing the command repeatedly has the effect of toggling focus between the last two windows that were active.

4.11.13 Minimize All

Menu: Window > Minimize All

Default Shortcut Key: none

Macro function: MinimizeAll()

The Minimize All command can be used to quickly minimize all open editor windows. Minimized windows are placed at the bottom of the *client area* in iconic form. The size and position of each window is stored so that any window can assume its former position once restored. Windows can be restored individually or with the Restore All command. To neatly arrange minimized icons which have become out of line, use the Arrange Icons command.

4.11.14 Restore All

Menu: Window > Restore All

Default Shortcut Key: none

Macro function: RestoreAll()

The Restore All command can be used to return all minimized windows to their former sizes and positions.

4.11.15 Maximize All

Menu: Window > Maximize All

Default Shortcut Key: none

Macro function: MaximizeAll()

The Maximize All command can be used to maximize all editor windows. The current window will consume the entire *client area*, and all other windows can be conceptualized as having been placed behind the current window.

When an editor window is maximized, its Minimize/Maximize/Close icon set is moved to the far right of the main menu bar. Once the window is minimized or restored, its icons are moved back to its title bar.

4.11.16 Close All

Menu: Window > Close All

Default Shortcut Key: none

Macro function: CloseAll()

The Close All command is used to close all open windows within the editor. If unsaved changes have been made to any file, a dialog box will appear for each such file to alert you to this fact. You will then be able to choose whether to save the changes before closing, close without saving, or cancel the Close All operation.

You can quickly tell whether a file has unsaved changes by looking for an asterisk (*) to the left of its name in the title bar or on its File Tab.

If you would prefer that Boxer be minimized automatically when the last file is closed, there is a checkbox on the Configure | Preferences | Other options page to achieve this. The option is titled *Minimize Boxer when closing last file*.

The Window | Close All command is functionally equivalent to the File | Close All command, since each file resides in its own window.

4.11.17 Close All but Active

Menu: Window > Close All but Active

Default Shortcut Key: none

Macro function: CloseAllButActive()

Use this command to close all open editing windows *except* the current window.

You can quickly tell whether a file has unsaved changes by looking for an asterisk (*) to the left of its name in the title bar or on its File Tab.

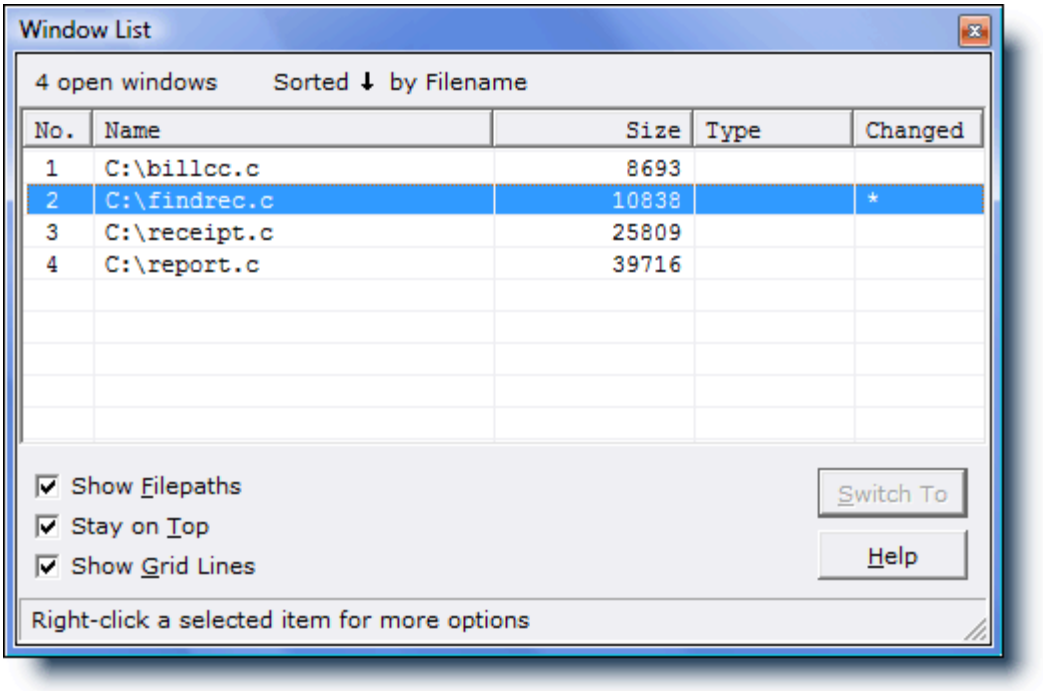
4.11.18 Window List

Menu: Window > Window List

Default Shortcut Key: none

Macro function: WindowList()

The Window List command presents a pop-up dialog containing the names of all currently open windows. A count of the windows is displayed at the top of the dialog.



Double-click or press *Enter* to make the selected window active. Right-clicking provides access to the Window List *context menu*. This menu provides options to Save, switch to, minimize, maximize, restore or close the selected window. The *Delete* key can be used to close the selected file.

The content of the Window List can be sorted by clicking on any of the field headers at the top of the listbox control. A second click on the same header reverses the order of the sort.

The Window List dialog is resizeable so it can accommodate long filepaths. The window is *non-modal* so that it can remain open while focus returns to an editor window.

Boxer's File Tabs also provide another method of switching among open windows, as do the Window Next and Window Previous commands.

If the Window List is left on-screen when Boxer is closed, it will be automatically reopened if the edit session is later restored.

4.12 Help Menu

4.12.1 Boxer Help

Menu: Help > Boxer Help

Default Shortcut Key: F1

Help is available at any time within Boxer by pressing F1. In most cases, the help topic presented will be sensitive to the context in which help was requested.

While cursoring within the main menu or a context menu Help will be presented for the highlighted menu item.

Most dialog boxes contain a *Help* button which presents the help topic dealing with that dialog box.

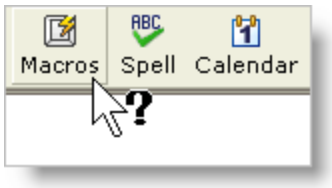
When editing a source code file for which Syntax Highlighting has been defined, the Help command can be used to summon language-specific help information for the word beneath the text cursor. The pathname of the help file which is associated with the language is defined in the *Help File* parameter of the Syntax Highlighting information for that language. See the Syntax Highlighting topic for more information about this capability.

4.12.2 Help On

Menu: Help > Help On

Default Shortcut Key: Shift+F1

The Help On command is used to activate a special mode in which the mouse cursor is changed to a help icon with an arrow cursor. In this mode the mouse is relieved of its conventional duties, and Help information will be displayed for the next object or menu item clicked upon.



The mouse arrow cursor will change to a 'no' cursor when atop an item for which help is not available, or not applicable.

4.12.3 FAQs

Menu: Help > FAQs

Default Shortcut Key: none

Frequently Asked Questions

After I order, will I get a password or key to convert the evaluation version of Boxer into a licensed version?
No. New software will be sent which is easily installed atop the evaluation version. All of your settings will be maintained. Software keys are frequently distributed on 'pirate' Internet sites, thus reducing sales and driving up the cost of software for paying customers.

Why can't other programs see the text I copied to the clipboard in Boxer?
You are almost certainly using an internal clipboard, rather than the Windows clipboard. Other programs can't see text that is placed on Boxer's internal clipboards. See the Set Clipboard command for details.

Why am I having trouble opening filenames from Explorer when they contain embedded spaces?
This is due to a bug in Explorer. It doesn't enclose a filename in double quotes before sending it off to the associated application. In the file associations set up by Boxer's installer, double quotes have been added around "%1", so you'll find these associations (.TXT, .BAT, etc) work fine. But for any file associations you create yourself, or if you elected not to allow Boxer's installer to create the associations, you'll need to manually edit the association to have double quotes around "%1". You'll find that Boxer's help topic entitled 'File Associations' has additional useful information about this subject.

Why can't I see all my Windows fonts in the Screen Font dialog?
Boxer requires that *fixed width* fonts (monospace fonts) be used, so the Screen Font dialog box does not display *proportionally spaced* fonts. This is required, in part, to ensure that columnar selections can be highlighted neatly in rectangular blocks, and so that the Column Ruler can be used. These features would not be possible if the use of proportionally spaced fonts was permitted.

Will there be a German version of Boxer for Windows, as there was for earlier Boxer products?
It appears unlikely. We learned from our earlier products that the effort to release a program in a new language is quite substantial. It appears that our resources can be better spent enhancing our current products, or developing new ones.

Will there ever be a Linux version of Boxer?
That's uncertain at this time. We're keeping an eye on the Linux market, and will continue to do so.

How long did it take to develop Boxer for Windows?
The initial development took almost two years. Boxer for Windows was a ground-up effort, with almost none of the code from our earlier products being used in its development.

What language was Boxer written in? How many lines of code? What development tool was used?

Boxer currently consists of over 110,000 lines of C++ code. Borland's C++ Builder was used for development.

Where did the name 'Boxer' come from?

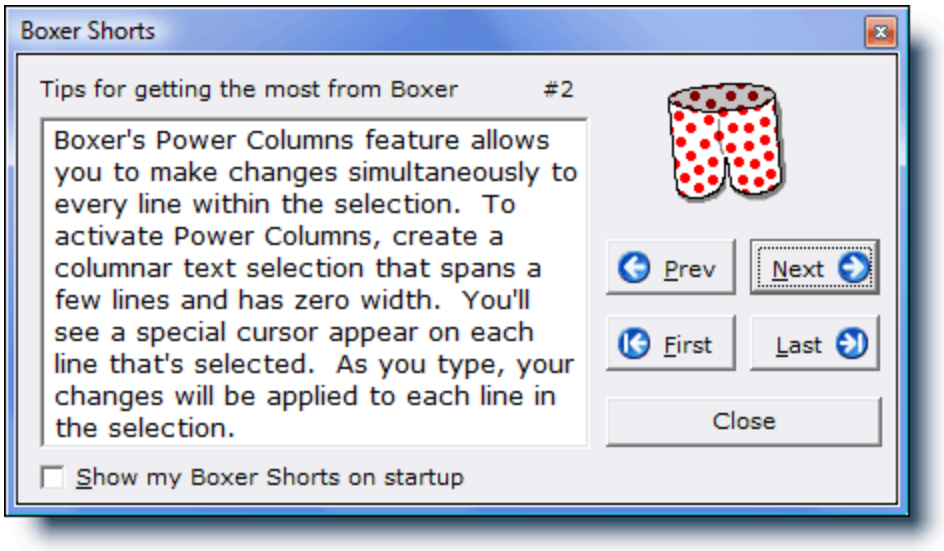
In the mid 1980's, one of the most popular editors for the PC was a product called BRIEF, which was then marketed by a company called UnderWare. In fact, the very first lines of Boxer/DOS were written using Brief, until Boxer was able to edit its own code. The name Boxer was simply a play on words: another style of men's underwear!

4.12.4 Boxer Shorts

Menu: Help > Boxer Shorts

Default Shortcut Key: none

'Boxer Shorts' are a collection of useful tips which are displayed in a popup window:



The tips presented in Boxer Shorts will help you to discover some of Boxer's lesser known features and capabilities. You can page through the tips with the buttons supplied, or select the *Remember to 'pull up' your Boxer Shorts upon startup* option so that the tips will be shown each time Boxer starts.

If you think you have a clever tip which could benefit other Boxer users, please send it to info@boxersoftware.com. We'll include the best tips we receive in future editions of Boxer Shorts.

4.12.5 Technical Support

Menu: Help > Technical Support

Default Shortcut Key: none

There are several ways to receive technical support for Boxer. The first and most obvious resource is the online Help. Online help contains detailed information on the configuration and use of Boxer, and for all of its commands. If you are having problems, please consult the relevant section of help before contacting us for support. You may also be able to find answers to some common questions on our website:

www.boxersoftware.com

Email
You can send electronic mail to us via the Internet. We prefer this method of support since it allows us to fully research a problem before responding. Also, we can sometimes reuse an earlier reply for a problem which has been experienced by more than one person. We typically check email several times a day:

support@boxersoftware.com

Telephone
You can also reach us by telephone Monday through Friday, 10:00 AM to 4:00 PM, Mountain Standard Time.

Voice: +1-602-485-1635

Postal Mail or Fax
Finally, you can mail or fax your inquiry to us. If you choose one of these methods, please be sure to describe your problem fully and include any information which may help us to diagnose the problem. Whenever possible, please provide an email address so that we can make return contact quickly and easily.

Fax: +1-602-485-1636

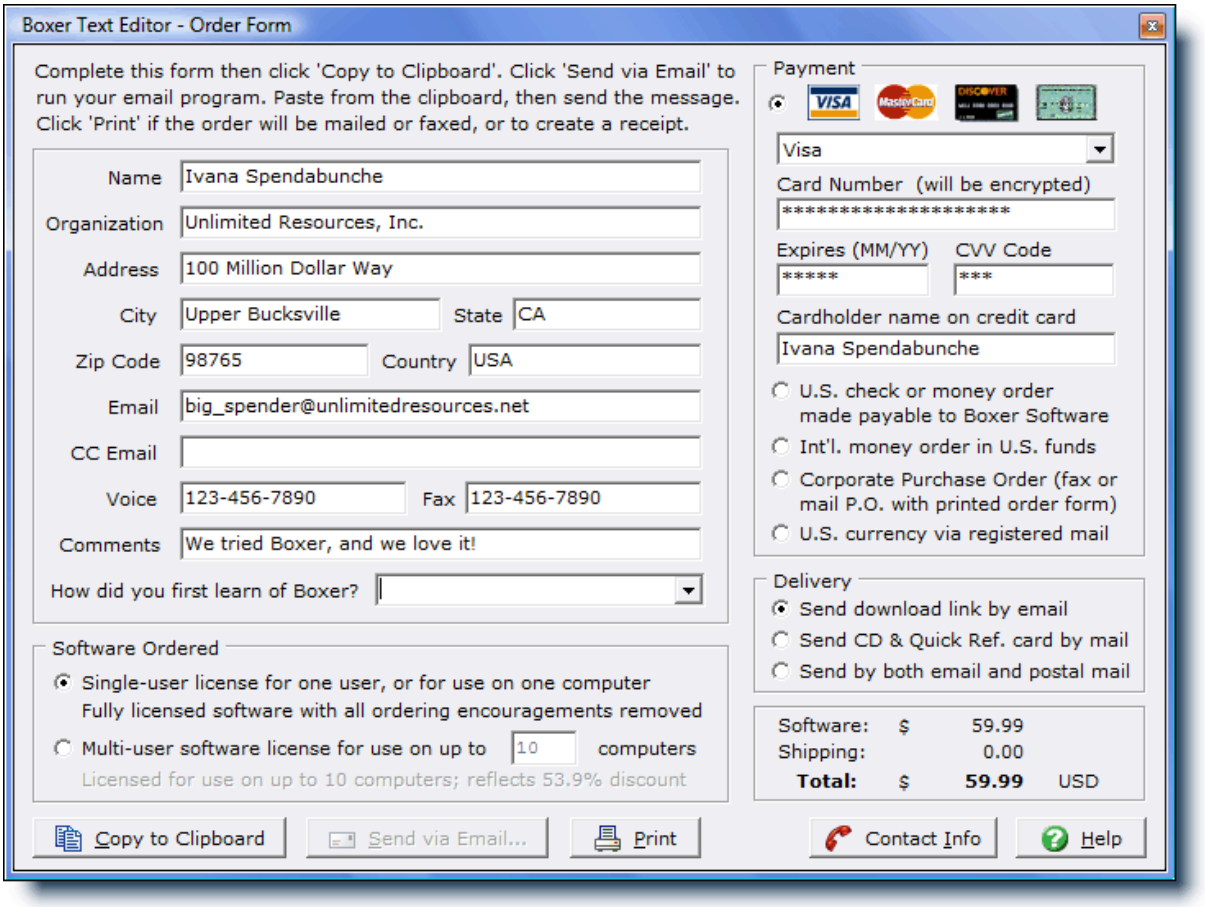
Boxer Software
PO Box 14545
Scottsdale, AZ
85267-4545 U.S.A.

4.12.6 Order Boxer

Menu: Help > Order Boxer

Default Shortcut Key: none

Boxer has an in-software order form to make ordering fast and easy. The order form is available by selecting the *Order Boxer* option from the Help menu, or by clicking the dollar bill icon on the toolbar of the evaluation version.



This order form can be used to submit your order by email, or to print an order form which can later be faxed or mailed along with payment. In all cases you can be assured that your order will receive prompt attention, and that we will safeguard your personal information. Boxer Software does not share its customers' mailing addresses, or email addresses, with any third parties.

☞ If you prefer to print an order form from within this help file, and then mail or fax it to us, use this order form.

The Order Form will compute your total automatically as you complete your order. If a Multi-User License is being ordered, click the appropriate option and enter the quantity desired. The total will be updated automatically to reflect the quantity ordered. Likewise, shipping is computed according to the destination country, and depending on whether delivery will be made by email or postal mail (delivery by email is free). If you elect to have the software sent via email, an http link will be sent from which you can download the software; the software is not sent by email attachment.

Ordering by Email

Complete the form, and then click *Copy to Clipboard* to copy the information entered to the Windows clipboard. Click *Send via Email* to launch your email program. The *To*

field of your email program should auto-fill with sales@boxersoftware.com. Paste the order information from the clipboard into the message body and send the message in the usual way. Note that your credit card information will be encoded using a proprietary encoding algorithm for added security. We will decode the information after your order arrives.

☞ If your email program does not launch after clicking *Send via Email*, simply start it in the usual way and paste the content on the clipboard into the body of a new message. Send the message to sales@boxersoftware.com.

Ordering at our Website

Visit www.boxersoftware.com to order from our secure order page. Full ordering details are provided at the site.

Ordering by Phone

Call toll-free within the U.S. and Canada at `1-800-98-BOXER (1-800-982-6937)` to order. Have your credit card ready; our sales representative will prompt you for the required information. From outside the U.S. and Canada call `+1-602-485-1635`. Business hours are Monday through Friday, 9 AM to 5 PM MST.

Ordering by Fax

Complete the form, and then click *Print*. Fax the order form to Boxer Software at `+1-602-485-1636`. Note that your credit card information will be encoded with a proprietary encoding algorithm for added security. We will decode the information after your order arrives. Our fax line is available 24 hours a day.

Ordering by Mail

Complete the form, and then click *Print*. Mail the order form to `Boxer Software, PO Box 14545, Scottsdale, AZ 85267-4545`. Note that your credit card information will be encoded with a proprietary encoding algorithm for added security. We will decode the information when your order arrives.

Ordering from Overseas

International Agents are available for those who might prefer to place their order with a local agent. Our agents accept payment in local currency and ship product from stock. Technical support services are also available.

Payment

Payment can be made in a variety of ways:

Credit Card
Visa, MasterCard, Discover or American Express

U.S. Check or Money Order
Made payable to 'Boxer Software'

Purchase Order
Purchase Orders can be mailed or faxed. Please make sure the Purchase Order includes both the shipping and invoicing addresses. Our payment terms are Net 30 days.

Western Union
Wire funds to 'David Hamel' and tell us the Control Number for the transaction, as well as the sender's name and the exact amount sent. Western Union also allows wire transfers to be made from their website: www.westernunion.com

U.S. Cash
Sent by certified or registered mail

PayPal
Send funds to 'sales@boxersoftware.com'. Don't have PayPal yet? Click here to sign up: www.paypal.com

International Money Order
Available at most banks. Money order should be drawn on a U.S. bank, in U.S. Funds, payable to 'Boxer Software'

International Postal Money Order
Available at the Post Office. Money order should be drawn in U.S. Funds, payable to 'Boxer Software'

Bank Wire Transfer
Please contact us for current bank transfer information. A $5.00 surcharge must be added to help offset the wire transfer fees we are assessed by our bank.
(**Note:** U.S. banks are not nearly as efficient as European banks with regard to bank wire transfers. Incoming transfers are slow, and receiving fees can be as high as $20.00. For this reason, we strongly encourage using another method of payment.)

4.12.7 Boxer Software Order Form

If you prefer not to order online, use the *Print* button to print this order form, and then mail or fax it to us.

BOXER SOFTWARE ORDER FORM

NAME: _____

(COMPANY:) _____

ADDRESS: _____ CITY/TOWN: _____

STATE/PROV: _____ ZIP/POST CODE: _____

COUNTRY: _____ PHONE: _____

EMAIL: _____

```
 MC/VISA/DISCOVER/AMEX: _____ Exp ___ /___

 CARDHOLDER: _____

 SIGNATURE:  _____ DATE: ____ /____ /____

HOW DID YOU FIRST LEARN OF BOXER SOFTWARE? _____

FOR WHAT TASKS DO YOU USE OUR PRODUCT?  _____

QTY  DESCRIPTION                                  PRICE   TOTAL

___  Boxer Text Editor                            $59.99 ____.__

___  Text Monkey                                   29.99 ____.__

___  The Permutator                                49.99 ____.__

___  File Append and Split Tool                    19.95 ____.__

___  Shipping: $4.00 U.S./Canada, $6.00 elsewhere       ____.__

                                            TOTAL:       ____.__

_____________________________________________________________________

  Boxer Software, Post Office Box 14545, Scottsdale, AZ 85267-4545
Sales: 800-982-6937   Voice: +1-602-485-1635  Fax  :
+1-602-485-1636
      sales@boxersoftware.com         www.boxersoftware.com
```

4.12.8 Check for Latest Version

Menu: Help > Check for Latest Version

Default Shortcut Key: none

This command can be used to check if the version of Boxer being used is still up-to-date. This command will launch your Internet browser to display a special page at the Boxer Software website. If a new version of Boxer is available, details will be given on how to get the update. Minor updates to Boxer will be made available free-of-charge. See the topic Upgrade Information for more information.

In order to launch your Internet browser, Boxer relies upon the operating system shell's ability to open an Internet address. When an Internet browser is installed, it typically establishes itself as the program which is called by the shell to open such addresses. This is true of all common browsers you are likely to encounter. If you find that your Internet browser is *not* launched by Boxer, or if some other inactive browser is launched instead, it's because your active browser has not established

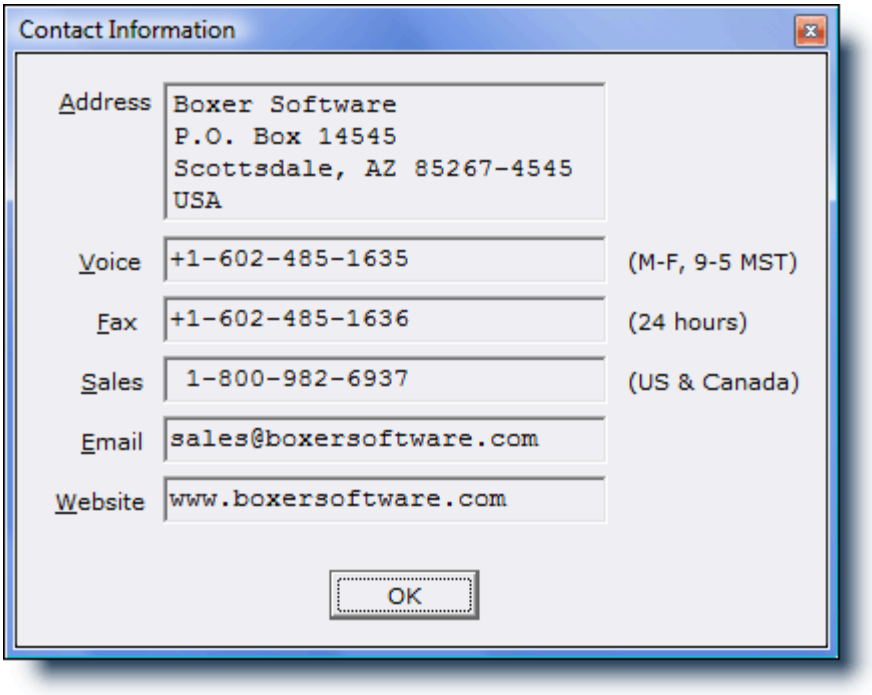
itself as the one that processes the 'open' request from the operating system shell for Internet addresses. This situation should be rare, cannot be remedied by Boxer, and is not due to any shortcomings in Boxer.

4.12.9 Contact Information

Menu: Help > Contact Information

Default Shortcut Key: none

The Contact Information command displays this dialog:

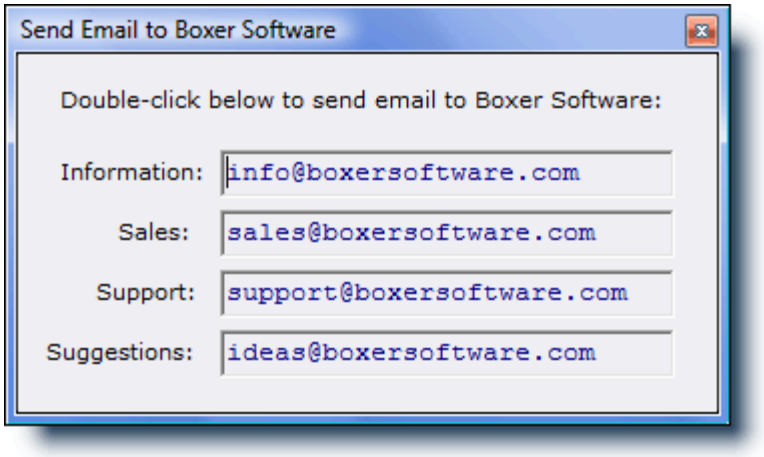


4.12.10 Email Boxer Software

Menu: Help > Email Boxer Software

Default Shortcut Key: none

This command displays a dialog box with email addresses which can be used to contact us at Boxer Software. Just click on the email address that fits your need, and your email client will be run so that a message can be sent.



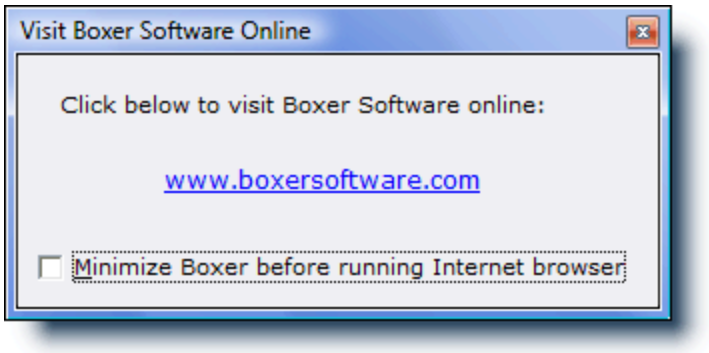
☞ In order to launch your email program, Boxer relies upon the operating system shell's ability to process a 'mailto' directive. When an email client program is installed, it typically establishes itself as the program which is called by the shell to process the 'mailto' directive. If you find that your active email program is not launched by Boxer, or if some other inactive email program is launched instead, it's probably because your active email program did not establish itself to be the program that processes mailto commands. This situation cannot be remedied by Boxer, and is not due to any shortcomings in Boxer. You might consult the documentation of your email program or contact its vendor.

4.12.11 Boxer Software Website

Menu: Help > Boxer Software Website

Default Shortcut Key: none

This command displays a dialog box with the address of the Boxer Software Website. Just click the address and your Internet browser will be run, taking you to our website. If you prefer that Boxer be minimized before the browser is run, an option is provided to do so.



The Boxer Software Website has a host of information that will be of interest to all users of Boxer. We invite you to visit the site periodically to learn about new products,

upgrades, and usage tips, and for other information which will be posted.

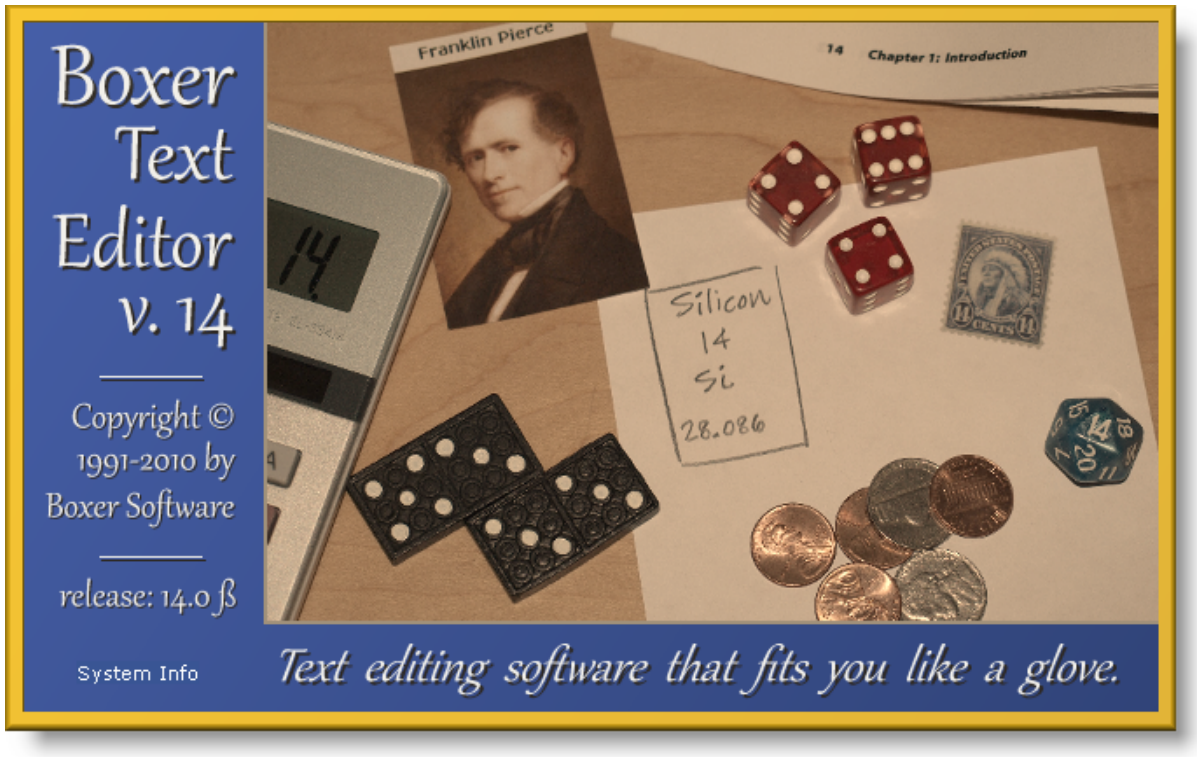
You can also visit our site by clicking here: www.boxersoftware.com

4.12.12 About Boxer

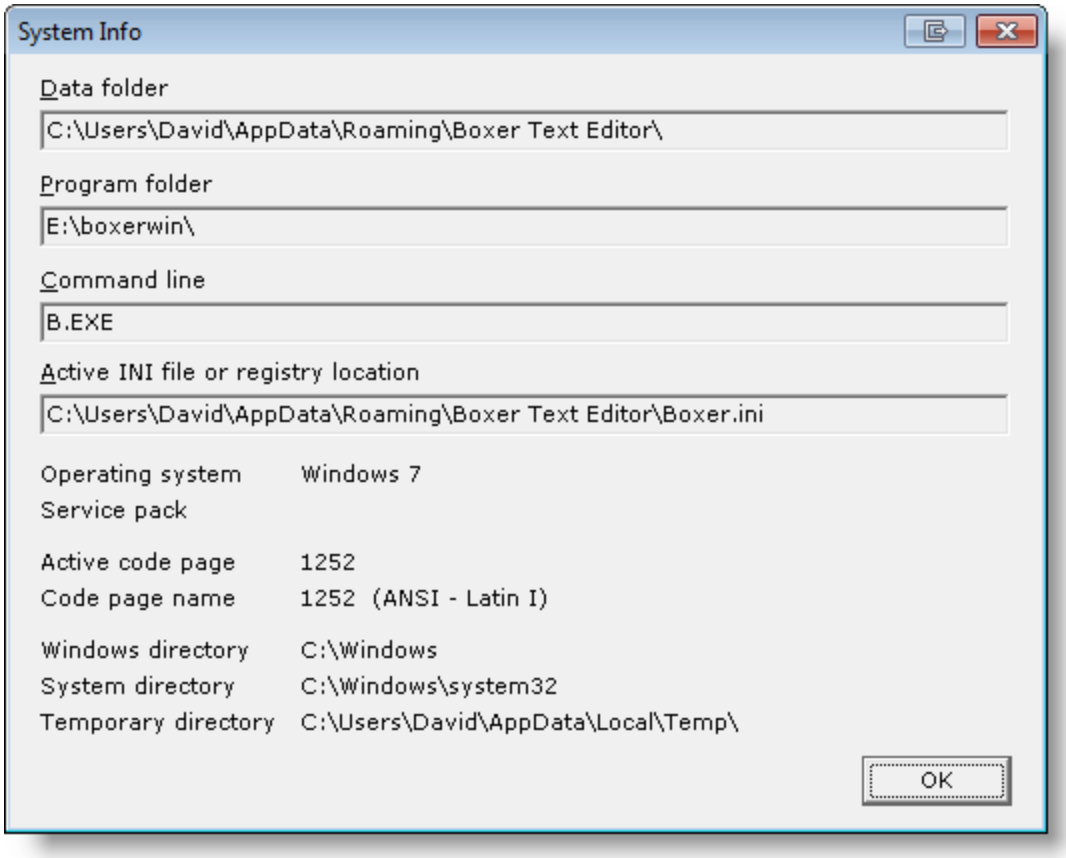
Menu: Help > About Boxer

Default Shortcut Key: none

The About Boxer command displays a popup box with information about the software version and copyright.



When the "System Info" link at the lower left of the image is clicked, this dialog is displayed:



Data folder
This edit box displays the full path of the current *Data folder*. You can use the Explore Data Folder command to open an Explorer window that points to this folder.

Program folder
This edit box displays the full path of the current *Program folder*. You can use the Explore Program Folder command to open an Explorer window that points to this folder.

Command line
This edit box displays the full command line that was used to invoke the current editing session. If any command line options have been used, they will appear in this display.

Active INI file or registry location
This edit box display the location from which the editor's settings were loaded, be it from a disk-based INI file, or from the *Windows registry*. If the -I command line option has been used to designate an alternate INI file, that directive's effect will be reflected here.

The various Boxer-related directories are presented in read-only edit boxes, making it possible to select them as text and copy them to the Windows clipboard.

5 Command Reference (alphabetically)

5.2 Active Spell Checking

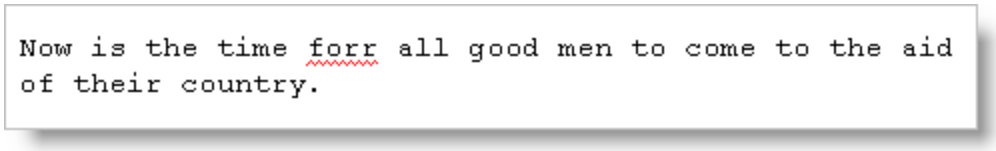
5.2 Active Spell Checking

Menu: View > Active Spell Checking

Default Shortcut Key: Alt+F7

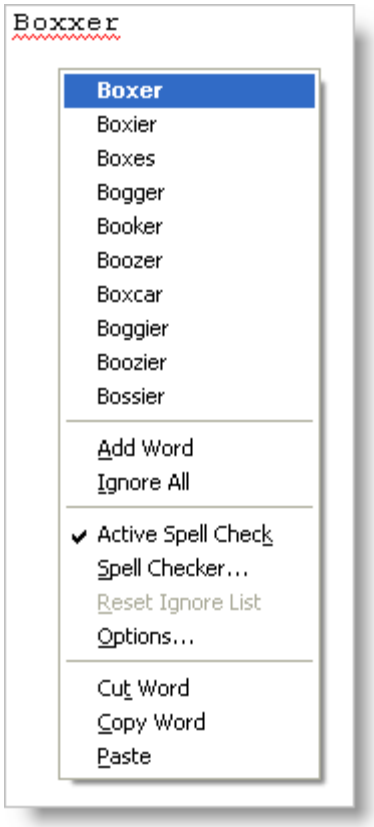
Macro function: ActiveSpellChecking()

The View Active Spell Checking command is used to toggle on/off a display mode in which misspelled words are underlined with a squiggly line. When this mode is on, Boxer will underline words as you type if they are not found in the active spell checker dictionary.



Note that Boxer does not wait for you to press *Space* and move off the word before deciding whether an underline will appear: the visual feedback is provided instantaneously. This makes it easier to experiment with alternative spellings, or to make a correction early in the word before it is completely typed.

If you right click on an underlined word, the misspelled word context menu will appear:



The context menu shows up to ten suggested corrections for the misspelled word, as well as other options related to spell checking.

Add Word
Use the *Add Word* option to add the offending word to the dictionary. Words which are added to the dictionary are saved within the file `userdict.txt` in Boxer's *data folder*. This file can also be edited within Boxer to add other words, or to remove words which may have been added mistakenly.

Words which are added to the user dictionary will be accepted as correctly spelled words in any case configuration in which they may occur. For example, if the word `ebay` is added to the dictionary, it will be accepted in any of the following forms: `eBay`, `ebAy`, and `ebaY`. This liberal processing was necessary because the third-party dictionary that Boxer uses is not processed in a case sensitive manner. Before this handling was put in place, the word `eBay` would always be reported as misspelled, even when `eBay` (or any variant) had been added to the user dictionary.

Ignore All
Use the *Ignore All* option to ignore the offending word, and to indicate that all other occurrences of the word should also be ignored.

Active Spell Check
Use this option to disable the Active Spell Checking feature. It can be reactivated using the View | Active Spell Checking command.

Spell Checker
Use this option to initiate a full spell checking scan with the Tools | Spell Checker command.

Reset Ignore List
This option can be used to clear the list of ignored words that have been added with the *Ignore All* option, or from earlier use of the *Ignore* button on the Spell Checker dialog.

Options
This menu entry opens the Configure | Preferences dialog to the page that contains the spell checker options.

Cut Word
Use this option to cut the misspelled word to the current clipboard.

Copy Word
Use this option to copy the misspelled word to the current clipboard.

Paste
Use this option to paste text from the current clipboard.

For general information about Boxer's spell checker, see the Spell Checker command.

The Configure | Preferences | Other dialog page contains a section with options related to Spell Checking.

5.3 Add All

Menu: Project > Add All

Default Shortcut Key: none

Macro function: ProjectAddAll()

Use the Add All command to add all open files to the active project.

This command can be used safely even when some of the open files are known to already reside within the active project. Duplicate entries will not result.

See the Project | New command for full details about Boxer's project file feature.

5.4 Add One

Menu: Project > Add One

Default Shortcut Key: none

Macro function: ProjectAddOne()

Use the Add One command to add the current file to the active project.

 See the Project | New command for full details about Boxer's project file feature.

5.5 Align Center

Menu: Paragraph > Align Center

Default Shortcut Key: Ctrl+F8

Macro function: AlignCenter()

The Align Center command centers the current line within the current Text Width. The cursor is moved to the line below upon completion.

If text is selected, all lines within the selected range will be centered.

This command will *not* cause words to be wrapped across lines. Use the Reformat command, with the desired Justification Style, for this purpose.

 If the Align Center command is issued when a columnar selection is in force, the effect of the command will be to center align the selected text within the extent of the rectangular selection.

5.6 Align Left

Menu: Paragraph > Align Left

Default Shortcut Key: Ctrl+F7

Macro function: AlignLeft()

The Align Left command moves the current line flush against the left edge, removing any indent which may have been present. The cursor is moved to the line below upon completion.

If text is selected, all lines within the selected range are affected.

This command will *not* cause words to be wrapped across lines. Use the Reformat command, with the desired Justification Style, for this purpose.

 If the Align Left command is issued when a columnar selection is in force, the effect of the command will be to left align the selected text within the extent of the rectangular selection.

5.7 Align Right

Menu: Paragraph > Align Right

Default Shortcut Key: Ctrl+F9

Macro function: AlignRight()

The Align Right command moves the current line flush against the right margin, as determined by the current Text Width. The cursor is moved to the line below upon completion.

If text is selected, all lines within the selected range are affected.

This command will *not* cause words to be wrapped across lines. Use the Reformat command, with the desired Justification Style, for this purpose.

If the Align Right command is issued when a columnar selection is in force, the effect of the command will be to right align the selected text within the extent of the rectangular selection.

5.8 Align Smooth

Menu: Paragraph > Align Smooth

Default Shortcut Key: Ctrl+F11

Macro function: AlignSmooth()

The Align Smooth command adjusts the current line to be flush against both the left and right margins. The right margin is determined according to the current Text Width. The cursor is moved to the line below upon completion.

Spaces are inserted alternately in the left, center and right portions of each line to minimize the appearance of *rivers and valleys* in the justified text.

If text is selected, all lines within the selected range are affected.

This command will *not* cause words to be wrapped across lines. Use the Reformat command, with the desired Justification Style, for this purpose.

If the Align Smooth command is issued when a columnar selection is in force, the effect of the command will be to smooth align the selected text within the extent of the rectangular selection.

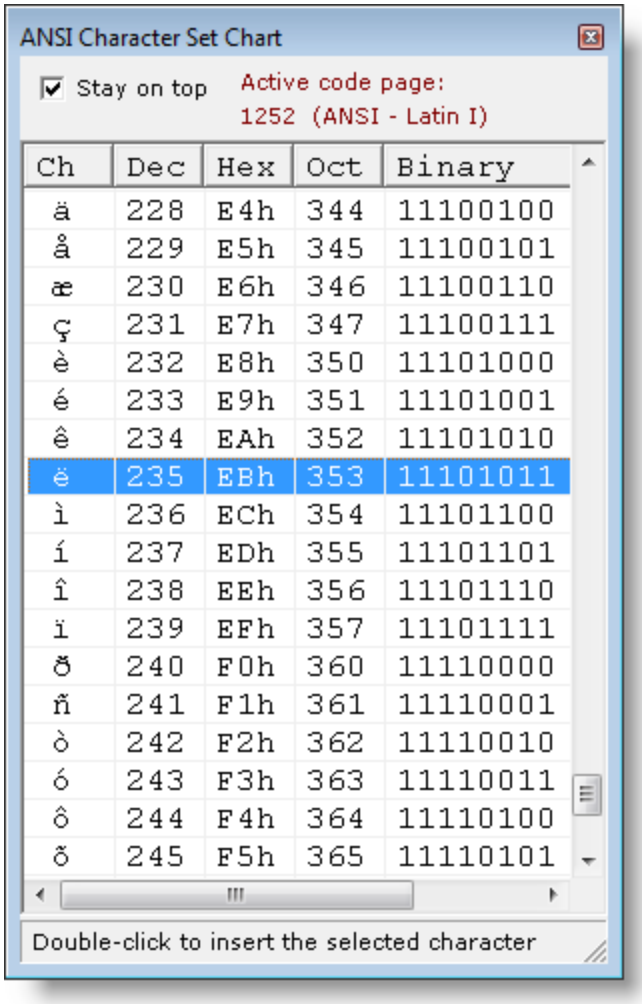
5.9 ANSI Chart

Menu: Tools > ANSI Chart

Default Shortcut Key: none

Macro function: ANSIChart()

The ANSI Chart command provides access to a popup chart which displays characters 0 to 255 in the ANSI character set. The character's visual representation is shown in the leftmost column, followed by the character value in *decimal*, *hexadecimal*, *octal* and *binary* formats. The active code page is also displayed at the top of the dialog:



Ch	Dec	Hex	Oct	Binary
ä	228	E4h	344	11100100
å	229	E5h	345	11100101
æ	230	E6h	346	11100110
ç	231	E7h	347	11100111
è	232	E8h	350	11101000
é	233	E9h	351	11101001
ê	234	EAh	352	11101010
ë	235	EBh	353	11101011
ì	236	ECh	354	11101100
í	237	EDh	355	11101101
î	238	EEh	356	11101110
ï	239	EFh	357	11101111
ð	240	F0h	360	11110000
ñ	241	F1h	361	11110001
ò	242	F2h	362	11110010
ó	243	F3h	363	11110011
ô	244	F4h	364	11110100
õ	245	F5h	365	11110101

To jump directly to a character of interest, simply press that character on the keyboard.

The ANSI Chart can be used to insert a character into the file being edited. Simply double click on the entry for the desired character or highlight the character in the chart and press *Enter*. When the need to insert a special character or symbol arises frequently, consider using the Insert Symbols feature rather than the ANSI Chart command. The Insert Symbols feature permits a defined character to be entered using a single keystroke.

Right-clicking on a selected item summons the ANSI Chart *context menu*. The context menu provides an option to copy the selected character to the current clipboard.

The ANSI Chart can also be used to convert between bases for values in the range 0 to 255. Simply locate the value to be converted in its proper column and read the converted value from the column of the new base.

If you prefer that the ANSI Chart remain atop other windows, select the *Stay on top* option. The ANSI Chart is a *non-modal* window, which allows it to remain on-screen after *focus* has been returned to another editing window.

 The ANSI Chart uses the same font and *code page* as the current screen font.

 If the ANSI Chart left on-screen when Boxer is closed, it will be automatically reopened if the edit session is later restored.

5.10 ANSI to OEM

Menu: Block > Convert Other > ANSI to OEM

Default Shortcut Key: none

Macro function: ANSItoOEM()

The ANSI to OEM command converts characters within the selected text from ANSI character encoding to OEM (ASCII) character encoding. These character encoding schemes share all the common alphabetic and numeric character mappings, but differ in the area of accented and/or graphic characters. A conversion may be appropriate when a file which was created with a Windows program must be prepared for use with a DOS program. Note that not all characters will have equivalents in the destination character set. In such cases a conversion will not be made for that character.

Boxer's ANSI Chart and OEM Chart commands can be useful for viewing the character assignments in each of these encoding schemes.

5.11 Append

Menu: Edit > Append

Default Shortcut Key: Shift+Ctrl+C

Macro function: Append()

The Append command copies the selected text from the current file and adds it to the current clipboard. The current clipboard might be the Windows clipboard or one of Boxer's eight internal clipboards. See the Edit | Set Clipboard command for details on changing the current clipboard.

If text is *not* selected, the Append command will operate on the current line. This

behavior can be controlled on the Configure | Preferences | Editing 1 options page. The option is titled *Cut/Copy/Append commands use current line when no text is selected*.

When a columnar selection (Block | Select Columnar) is placed on the clipboard, any under-length lines within the selected range will be extended with Spaces to match the width of the rectangular text block. This ensures that the block will behave as expected if the Paste command is later used to insert the text at a new location. Likewise, any Tab characters within the selected region will be converted to Spaces before being placed on the clipboard.

Text cannot be appended to a clipboard if the selection type (stream or columnar) differs from the type of the text which is already present on the clipboard.

When placing columnar text onto a clipboard, Boxer must take care so that subsequent Paste operations of that text will be performed properly. Columnar clipboard text must be pasted differently than stream text, since all lines must move rightward by the width of the text block. Notwithstanding this fact, columnar clipboard text placed onto the Windows clipboard by Boxer can still be pasted into other Windows applications. Boxer does not use a *private clipboard format* for this purpose.

Boxer's clipboard commands will sense the type of text data being placed on the Windows clipboard and, when appropriate, use a more descriptive clipboard format to tag that data. For example, when Boxer senses that HTML code is being copied to the clipboard, the text will also be placed in the HTML clipboard format. This allows a conforming program to paste the data more intelligently. Boxer will also tag Rich Text data (RTF) and Comma-Separated Value (CSV).

5.12 Apply Highlighting

Menu: View > Apply Highlighting

Default Shortcut Key: none

Macro function: ApplyHighlighting()

This command can be used to add the word at the text cursor -- or the currently selected text -- to the list of phrases that are to be highlighted by the Text Highlighting command. This feature can be used to make table headings stand out, or to add emphasis to any class of words or phrases that might be desired. The highlighting strings are saved and restored from session to session. The color used to highlight the designated strings is configurable using the Configure | Colors dialog. Text Highlighting can be applied to normal text files, or to program files which are already being Syntax Highlighted. The highlighting of strings can be quickly toggled on/off by using the View | Text Highlighting command.

This command is also available from the context menu.

5.13 Arrange Icons

Menu: Window > Arrange Icons

Default Shortcut Key: none

Macro function: ArrangeIcons()

The Arrange Icons command can be used to neatly position minimized windows icons at the bottom of the *client area*. When several windows have been minimized, and some are later restored to normal size, the display of the remaining minimized windows can become untidy. The Arrange Icons command is useful in this situation.

5.14 ASCII to EBCDIC

Menu: Block > Convert Other > ASCII to EBCDIC

Default Shortcut Key: none

Macro function: ASCIItoEBCDIC()

This command will convert text encoded in the ASCII character set to the EBCDIC character set. The text to be converted must first be selected. If an entire file is to be converted, use the Select All Text command to select the whole file.

EBCDIC is a character encoding system used primarily on mainframe computers. The ASCII character encoding system is used widely on personal computers. At times, a file that uses ASCII encoding may need to be converted for use on a computer that uses the EBCDIC character encoding system. This command can be used for that purpose.

☞ ASCII is an acronym for American Standard Code for Information Interchange.

☞ EBCDIC is an acronym for Extended Binary Coded Decimal Interchange Code.

5.15 Auto-Complete

Menu: Tools > Auto-Complete

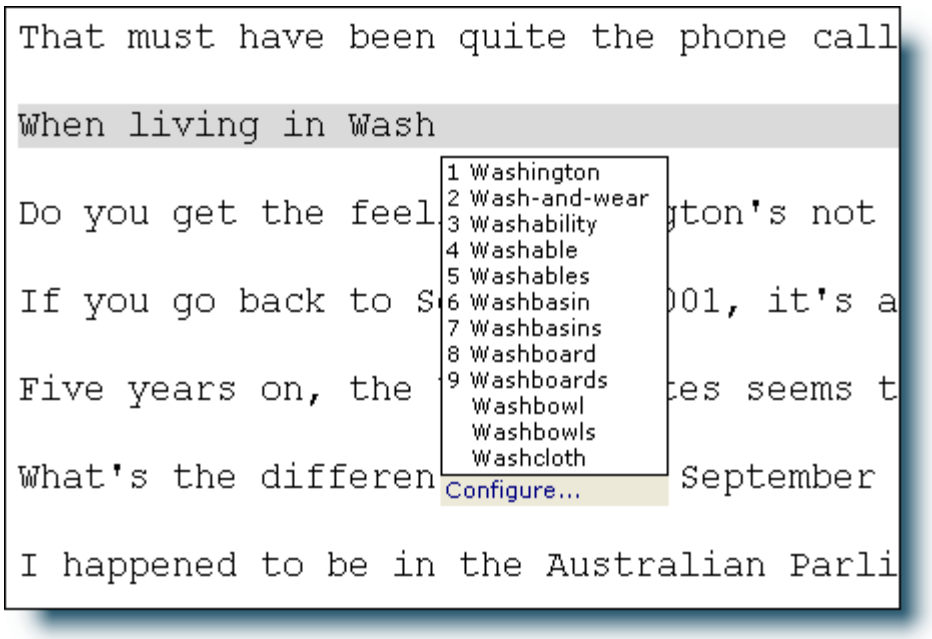
Default Shortcut Key: Ctrl+Space

Macro function: none

Boxer's Auto-Complete feature will display a popup list of matching words that can be used to complete a partially typed word at the text cursor. The popup list will appear after a (user-configurable) number of characters have been entered, and contains only those words that match the characters just typed. There is no need to interact with the

list: if you prefer to keep typing, the list will simply disappear when it becomes irrelevant.

To select a word from the popup list, simply press the number displayed to its left, or use the down arrow to cursor into the list and press *Enter* at the desired entry.



The display (and recognition) of the hot numbers within the popup list is optional. It can be controlled on the Configure | Auto-Complete | Popup List dialog page.

If the suggestions within the popup list contain digits, the hot numbers feature will be automatically disabled. Otherwise, the entry of alphanumeric text strings would become confusing and error prone.

Discussion

Much of the utility of the Auto-Complete feature derives from the relevance of the words that are presented in the popup list.
Auto-Complete builds its list of matching words from four sources:

- user-defined phrases
- the existing text of the current document
- the reserved word list for the current file type (when applicable)
- an external, user-editable dictionary of 37,000 long words

User-defined phrases are those words and phrases that have been pre-defined on the Configure Auto-Complete - User-Defined dialog page. These might include common text strings that you use in your work, your mailing address, email address, etc. These phrases are given highest priority and will appear first in the completion list.
User-defined phrases can also be set to expand 1) as soon as they're typed (bte could

auto-expand to 'Boxer Text Editor'), 2) only when a delimiter is typed, or 3) only when the Trigger Key is pressed. See Configure Auto-Complete - User-Defined for full details.

Auto-Complete also analyzes the content of the current file to find potential matches for the completion list. This means that words and phrases that are particular to your document are automatically suggested as matching words. If you're editing a document that contains the term 'diethylthiocarbamate', that word will appear in the completion list, even though it doesn't appear in the dictionary proper. Auto-Complete also intelligently harvests phrases from program source code, making the feature useful for programmers as well.

When editing a file for which Syntax Highlighting has been defined, the reserved words for that language will be used as completion words.

Finally, Auto-Complete uses a large dictionary of English* words from which all shorter words have been removed. This dictionary is maintained in simple, uncompressed ASCII text format, and is therefore user-editable. The dictionary can be viewed and/or edited from the Configure Auto-Complete - Dictionary dialog page.

* Dictionaries for other languages are not available at this time, but if you can locate a large word list for the language of interest, you can use that list to replace the `AC_Words.txt` file provided with Boxer. The Sort Lines command has an option to sort lines by line length, making it easy to locate and remove smaller words from the list. If you assemble your list from multiple sources, use the Delete Duplicate Lines to remove duplicates. The final word list should contain one word/phrase per line, and be ASCII-sorted, case insensitive.

Customization

Auto-Complete is the type of feature that some users will have very strong feelings about. Users are sure to have different ideas about *when* the popup list should appear, *where* it should appear, *whether* it should appear at all, *how many* entries it should have, *which* words should appear in the list, etc. Auto-Complete has an **extensive** collection of configuration options to control every aspect of its operation. The following help topics cover the configuration of the Auto-Complete feature:

General Settings
Popup List Settings
User-Defined Phrases
Harvested Words
Dictionary Words
Excluded Words

If the default Auto-Complete settings don't feel perfect to you, you are encouraged to spend a few minutes experimenting with the various options to make sure it feels just right.

Manual Operation

Some users may find the popup list distracting, and opt to disable it. When Auto-Complete is configured not to display a popup list, the *Trigger Key* can be used to complete the partially typed word at the text cursor. Pressing the Trigger Key

repeatedly cycles through the available matches. By default, the Trigger Key for Auto-Compete is *Ctrl+Space*.

The popup list can be displayed at any time by using the Auto-Complete List command. This command is useful when the popup list has been disabled, or when you want to force the list to appear in a situation where it would not naturally appear (for example, when you haven't typed enough characters for it to appear).

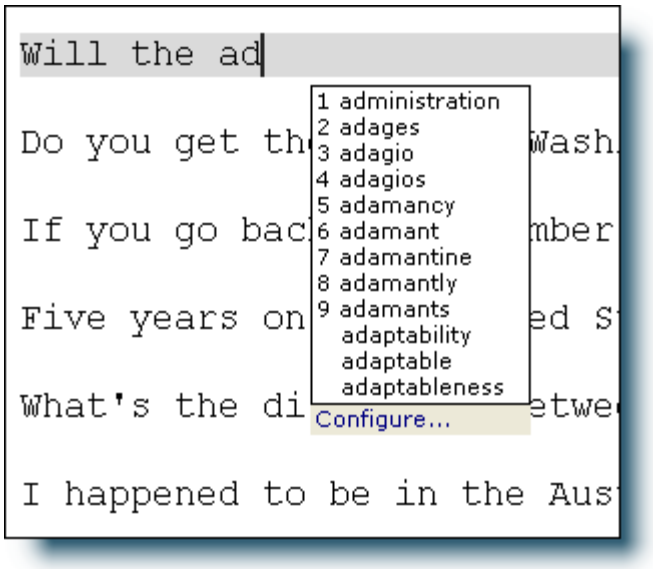
5.16 Auto-Complete List

Menu: Tools > Auto-Complete List

Default Shortcut Key: Ctrl+Alt+Down

Macro function: none

The Auto-Complete List command can be used to force the popup list of matching words to appear in situations when it wouldn't appear naturally on its own. One such case is when an insufficient number of characters has been typed to cause the list to appear. Another case is when the popup list has been configured not to appear at all.



For more complete information about Auto-Complete, see the Auto-Complete topic.

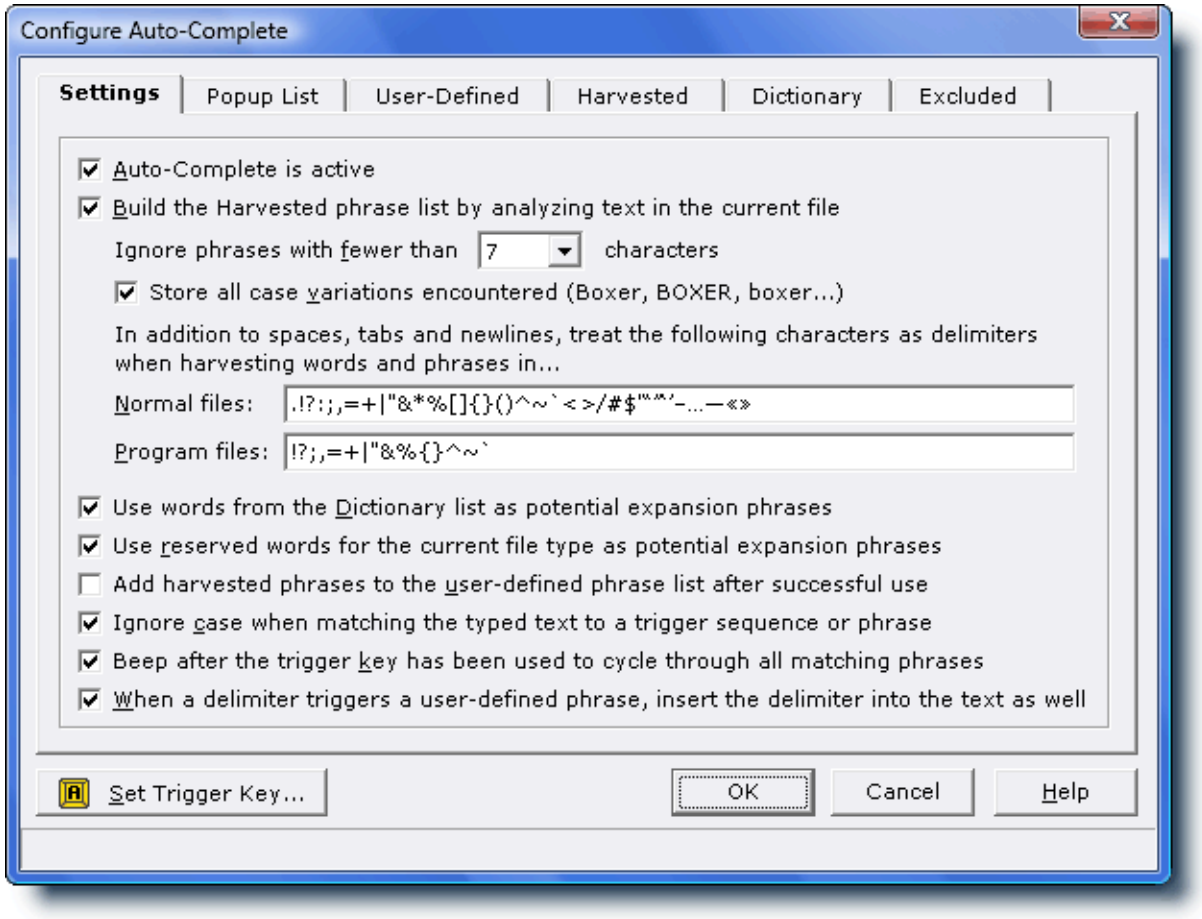
5.17 Auto-Complete - Settings

Menu: Configure > Auto-Complete

Default Shortcut Key: none

Macro function: none

The Settings tab of the Configure Auto-Complete dialog contains general options that relate to the Auto-Complete feature.



Auto-Complete is active
The checkbox is the master on/off switch for the Auto-Complete feature. If this box is unchecked, all Auto-Complete features and functions will be disabled.

Build the Harvested phrase list by analyzing text in the current file
Use this checkbox to control whether or not the current file will be analyzed to harvest words for use with the Auto-Complete feature. If harvesting is not performed, the matching word list will be built from other sources: user-defined phrases, reserved words, and the master dictionary.

Ignore phrases with fewer than *n* characters
Use this option to specify the shortest word that should be collected during harvesting. Auto-Complete is most effective when used to complete longer words, so there's little benefit to storing very small words when harvesting.

Store all case variations encountered (Boxer, BOXER, boxer...)
Use this checkbox to specify how case variations should be handled during

harvesting. When checked, all different case variations will be collected. When unchecked, only the first case variation of a given word will be stored.

In addition to spaces, tabs and newlines, treat the following characters as delimiters when harvesting words and phrases in...

Normal files
These characters will be treated as word delimiters when harvesting words from normal files -- that is, from files that do not have a Syntax Highlighting definition entry.

Program files
These characters will be treated as word delimiters when harvesting words from program files -- that is, from files that **do** have a Syntax Highlighting definition entry.


The *Program files* delimiter characters control how the Auto-Complete feature harvests code fragments from the current file. This list of delimiters will typically have fewer characters in it than the *Normal files* list of delimiters, so that longer code fragments can be collected. Regardless of the delimiters listed here, Auto-Complete will also harvest program files with a more restrictive delimiter list so that the individual elements of a code fragment appear on their own. For example, the code fragment `structure_name->variable_name[index_variable]` might be collected in its entirety due to the *Program files* delimiters that are in use. *Regardless* of those delimiters, `structure_name`, `variable_name` and `index_variable` will be harvested as individual entries as well.

Use words from the Dictionary list as potential expansion phrases
This checkbox controls whether or not dictionary words will be used to build the matching word list.

Use reserved words for the current file type as potential expansion phrases
This checkbox controls whether or not reserved words from the Syntax Highlighting information for the current file will be used to build the matching word list.

Add harvested phrases to the user-defined phrase list after successful use
When this option is checked, harvested words will be automatically added to the User-Defined phrase list after they have been used successfully in a word completion. In this way, the User-Defined phrase list will become filled with the words that you use most often, and the Auto-Complete feature will become more tuned to your work style.

Ignore case when matching the typed text to a trigger sequence or phrase
Use this option to indicate whether or not character case should be considered when comparing typed text to User-Defined trigger sequences and phrases.

Beep after the trigger key has been used to cycle through all matching phrases
This option controls whether or not an audible beep should occur after the Trigger Key has been used to cycle through all potential matches.

When a delimiter triggers a user-defined phrase, insert the delimiter into the

text as well
This option controls how a delimiter character should be treated when it is used to trigger a User-Defined phrase.

Set Trigger Key
Use the *Set Trigger Key* button to change the key that is used to complete a partially typed word, or expand a User-Defined phrase with trigger style activation.

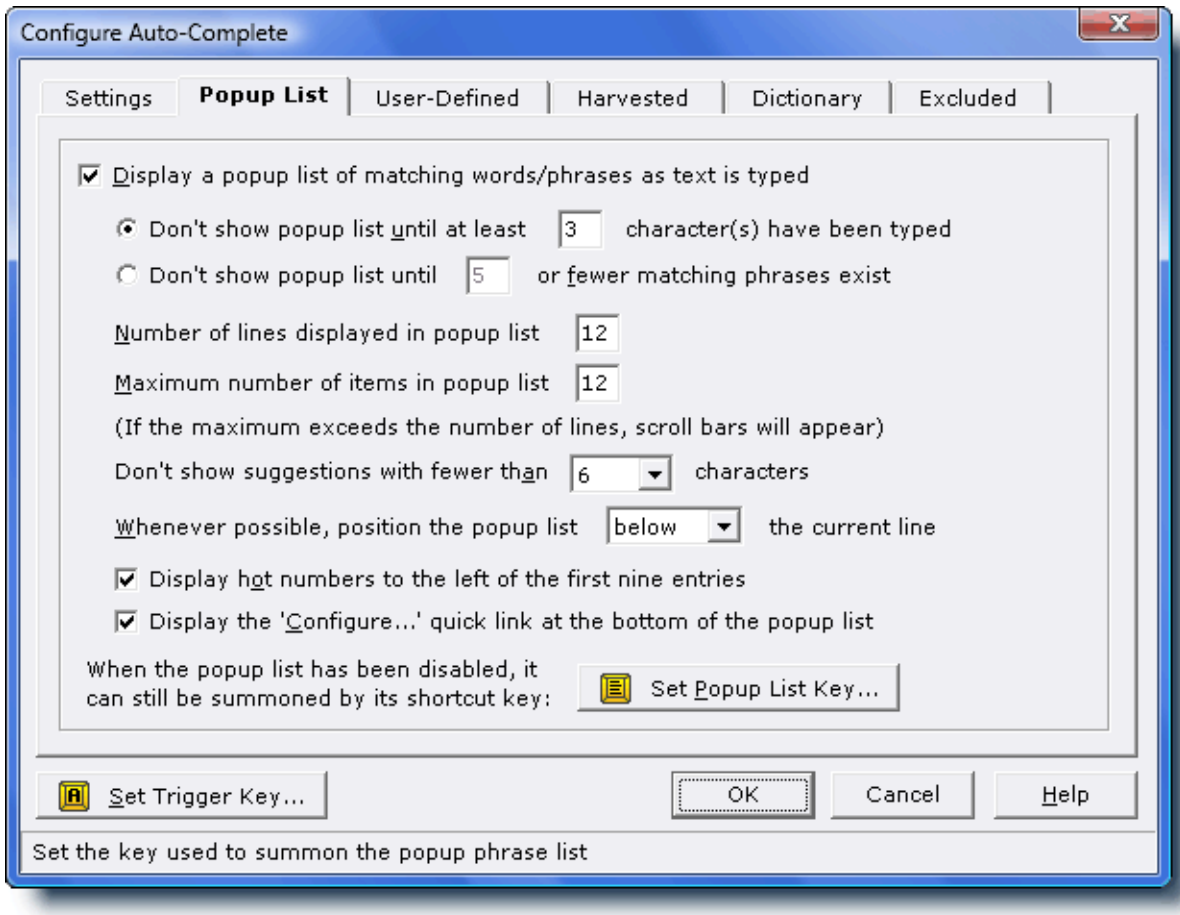
5.18 Auto-Complete - Popup List

Menu: Configure > Auto-Complete

Default Shortcut Key: none

Macro function: none

The Popup List tab of the Configure Auto-Complete dialog contains options that relate to the presentation of the popup list of matching words.



Display a popup list of matching words/phrases as text is typed
Use this checkbox to indicate whether or not the popup list box should appear while text is being typed. When disabled, the popup list can still be displayed using the Auto-Complete List command.

Don't show popup list until at least *n* characters have been typed
Use this option if you prefer that the display of the list be determined by the number of characters that have been typed. When this option is used, the popup list will appear as soon as '*n*' characters have been typed, provided there are matches available.

Don't show popup list until *n* or fewer matching phrases exist
Use this option if you prefer that the popup list not be shown until the number of matches falls *below* a certain threshold. When this option is used, display of the popup list will be suppressed until enough typing has occurred to narrow the list to a given number of matching words.

Number of lines displayed in popup list
This option controls the number of visible entries in the popup list -- ie, the height of the list.

Maximum number of items in popup list
This option controls the maximum number of items that will appear in the list. If this values exceed the number of lines, a vertical scroll bar will be added to the list.

Don't show suggestions with fewer than *n* characters
Use this option to specify the shortest word that should appear in the popup list.

Whenever possible, position the popup list above/below the current line
Use this option to specify where the popup list will appear in relation to the current line. When the current line is near screen top or screen bottom, the list may need to be moved to keep it on-screen.

Display hot numbers to the left of the first nine entries
This option is used to control the display and recognition of hot numbers within the popup-list.

Display the "Configure..." quick link at the bottom of the popup list
This option is used to control the display of the hot link at the bottom of the list.

Set Popup List Key
Use this button to change the key used to force display of the popup list. See the Auto-Complete List command for full details.

Set Trigger Key
Use the *Set Trigger Key* button to change the key that is used to complete a partially typed word, or expand a User-Defined phrase with trigger style activation.

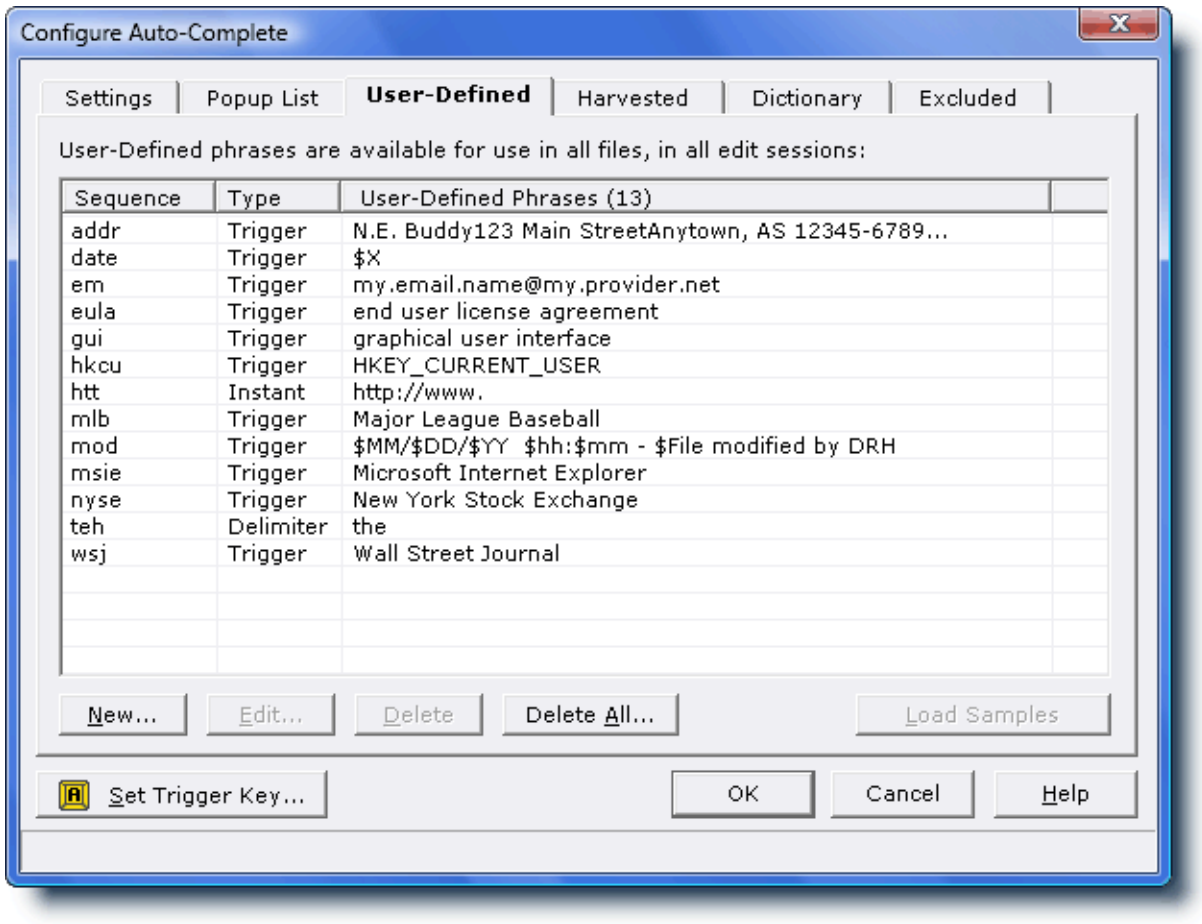
5.19 Auto-Complete - User-Defined

Menu: Configure > Auto-Complete

Default Shortcut Key: none

Macro function: none

The User-Defined tab of the Configure Auto-Complete dialog allows words and phrases to be defined for use with the Auto-Complete feature.



Discussion

One of the most powerful elements of Boxer's Auto-Complete function is the ability to define words and phrases which will expand automatically, after a delimiter character is typed, or when the *Trigger Key* is typed. The phrases in the dialog pictured above provide a good sampling of the types of assignments that are possible.

Three types of user-defined phrases are available:

- Instant - these phrases will be auto-inserted the instant the sequence string is typed.

In the sample phrase set, typing `'htt'` will cause the string `'http://www.'` to be inserted.

- Trigger - these phrases will be inserted only when the sequence string is typed followed by the Trigger Key. In the sample phrase set, if `'eula'` is typed, and the Trigger Key is typed, `'end user license agreement'` will be inserted.

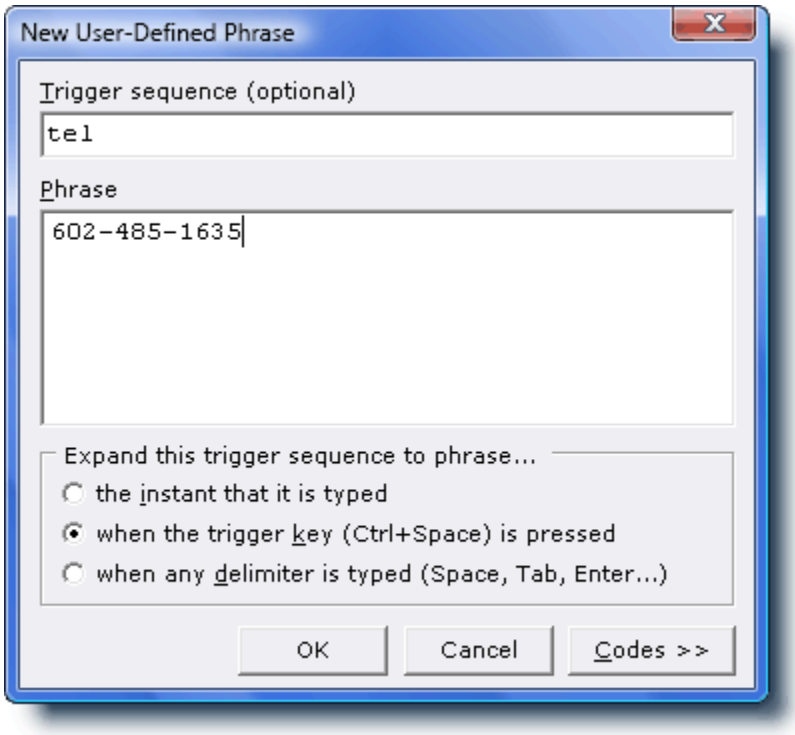
- Delimiter - these phrases will be auto-inserted only when the sequence string is followed by a delimiter character. In the sample phrase set, typing `'teh'` followed by (say) space will cause `'the'` to be inserted -- an auto-correction for mistyping `'the'`.

These three activation styles provide both utility and flexibility. Some phrases are best defined as *Instant*, while others are naturally more suitable to being *Trigger* or *Delimiter* style phrases.

You might want to invent your own conventions for defining phrases. By using an obscure lead-in or trailing character in the sequence string, virtually all phrases can be defined as *Instant*. For example, if the `addr` sequence string had been defined instead as `~addr`, its activation type could have been *Instant* since there would be almost no chance of `~addr` being typed in the course of normal work. Likewise, use `addr~` as the sequence string effectively makes ~ the new *Trigger Key*.

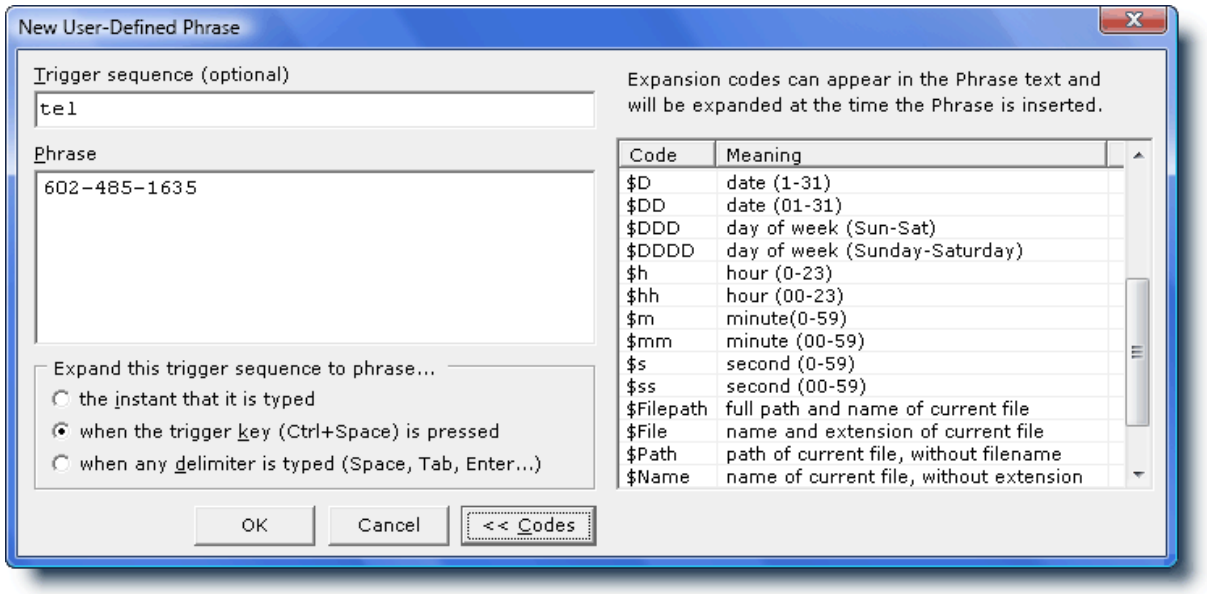
New

Use the New button to create a new User-Defined phrase. The following dialog will appear:



If the phrase will be triggered by a text string, enter the sequence string in the upper edit box. Enter the phrase itself in the *Phrase* memo box. Multi-line expansion phrases *are* allowed; press *Enter* to being a new line in the phrase. Finally, select the type of activation desired using the radiobuttons at the bottom of the dialog.

Expansion Codes
To define a phrase that includes *expansion codes*, click the *Codes* button to expose the list of expansion codes:



Expansion codes will be expanded when the phrase is inserted to reflect their meaning. A variety of codes for time, date and various filename functions are available.

Edit
Use the *Edit* button to edit the settings for the selected phrase.

Delete
Use the *Delete* button to delete the selected phrase.

Delete All
Use the *Delete All* button to erase all user-defined phrases. A confirmation will be required before the operation is performed.

Load Samples
The *Load Samples* button will add a small collection of sample phrases to the list of user-defined phrases. Any phrases that are already present in the list will not be disturbed.

Set Trigger Key
Use the *Set Trigger Key* button to change the key that is used to complete a partially typed word, or expand a User-Defined phrase with trigger style activation.

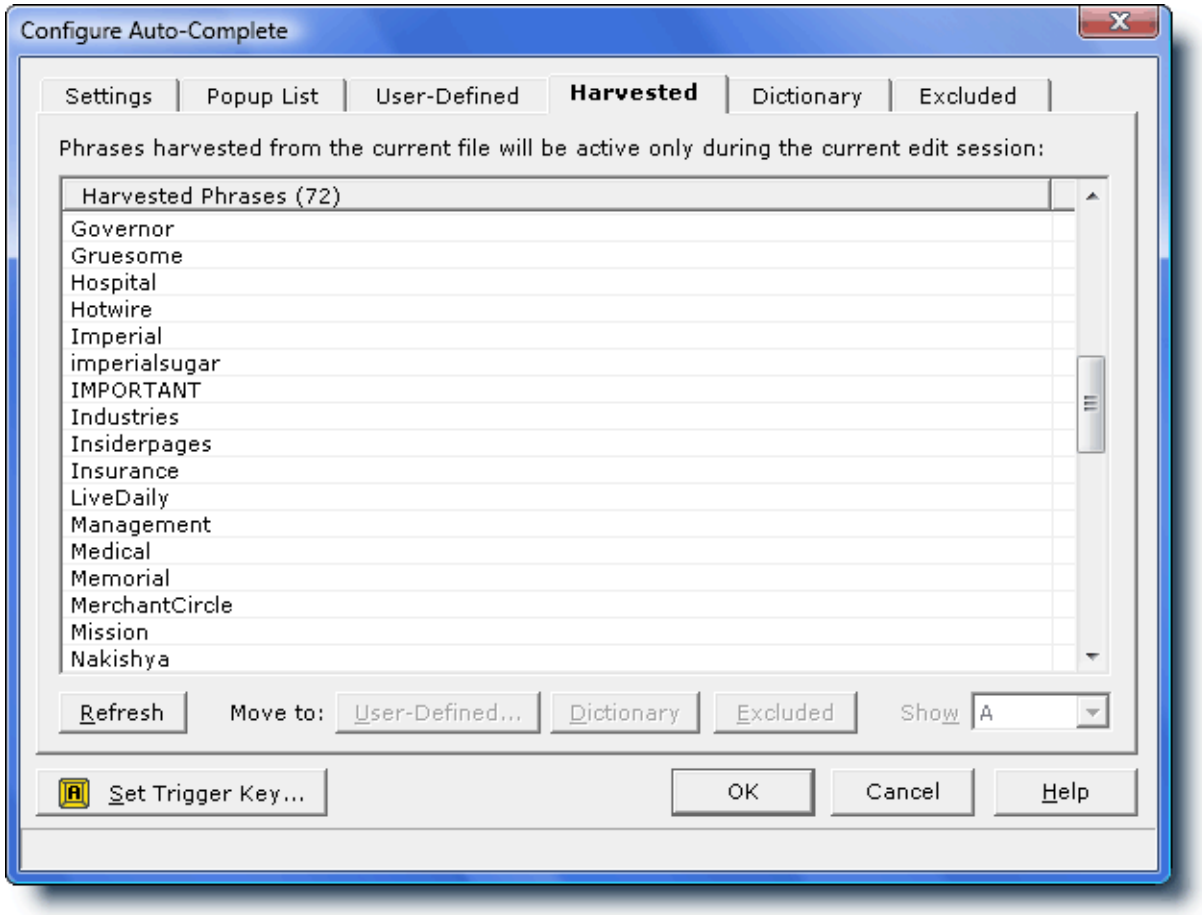
5.20 Auto-Complete - Harvested

Menu: Configure > Auto-Complete

Default Shortcut Key: none

Macro function: none

The Harvested tab of the Configure Auto-Complete dialog displays the words that have been harvested from the current document.



If the Harvested word list is large, the dialog will be displayed initially with an empty list. Use the *Show* option at the lower right to select the starting letter of the words you would like to view.

The Harvested word list is a temporary and transient word list. It is built on-the-fly by analyzing the current file. When an edit session ends, the Harvested word list is deleted. If the Harvested word list contains words that you would like to make permanent, there are buttons available to move words to other lists. It is not meaningful to delete a word from the Harvested word list, because it will simply reappear in the list the next time the file is next analyzed.

Refresh
Use the Refresh button to request that the current file be re-analyzed to build the harvested word list. This option is useful to check the effect of changes made on the Settings dialog tab.

Move to User-Defined
Use the *User-Defined* button to move the selected word to the User-Defined word list. The *New User-Defined Word* dialog will appear so that additional information can be provided.

Move to Dictionary
Use the *Dictionary* button to move the selected word to the master Dictionary. This ensures that the word will subsequently appear in the popup list of matching words, even if it does not already exist in the current file.

Move to Excluded
Use the *Excluded* button to move the selected word to the Excluded word list. Words in the Excluded word list will never appear in the popup list of matching words.

Set Trigger Key
Use the *Set Trigger Key* button to change the key that is used to complete a partially typed word, or expand a User-Defined phrase with trigger style activation.

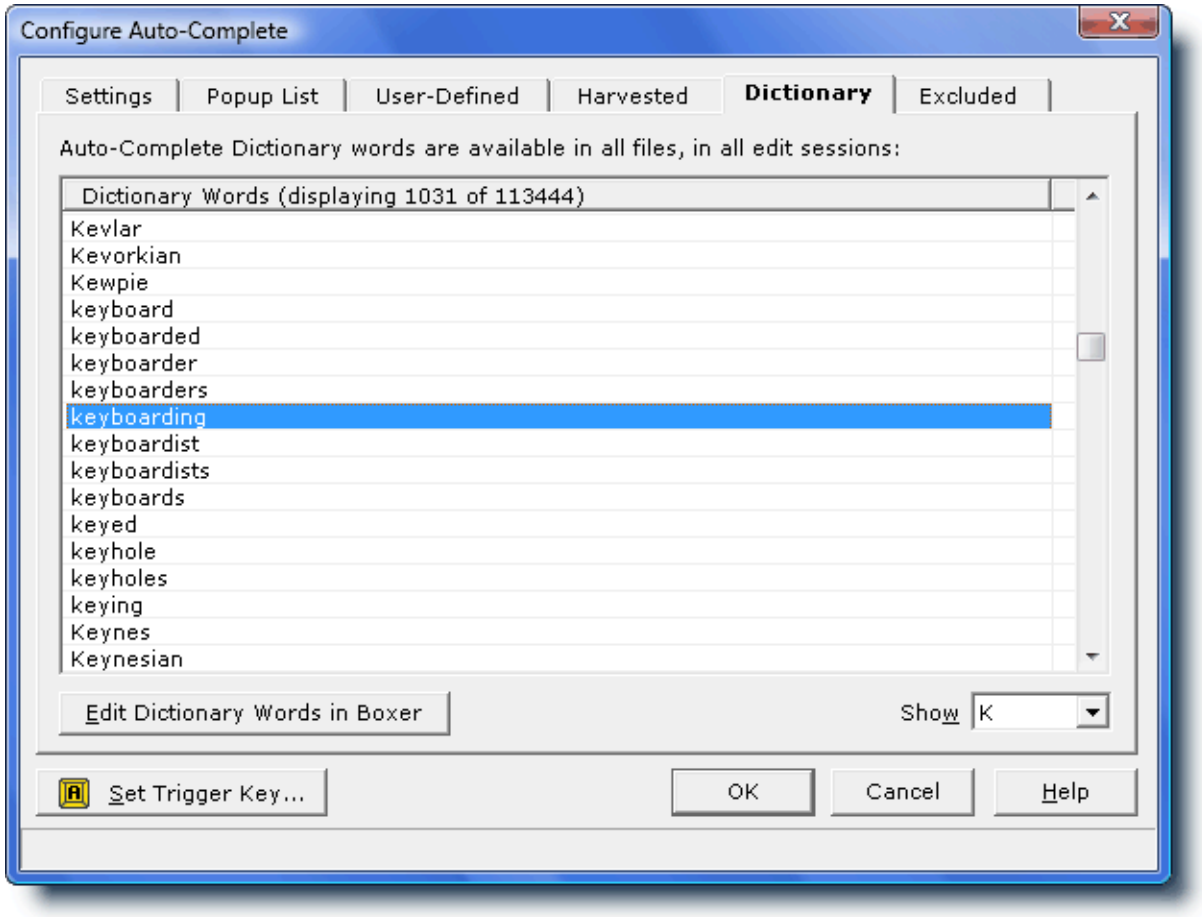
5.21 Auto-Complete - Dictionary

Menu: Configure > Auto-Complete

Default Shortcut Key: none

Macro function: none

The Dictionary tab of the Configure Auto-Complete dialog can be used to view or edit the Auto-Complete master word list.



Initially, the dialog will be displayed with an empty list. Use the *Show* option at the lower right to select the starting letter of the words you would like to view.

Edit Dictionary Words in Boxer
Click this button to load the dictionary word list into Boxer for viewing or editing. The word list is maintain in a simple ASCII text file, so it can be edited directly without complication. You can add new words, or remove existing words.

The format of the dictionary file is straightforward: one word per line, alphabetically sorted, case insensitive. You can use the Sort Lines command, if needed to sort the file after additions have been made.

Dictionaries for other languages are not available at this time, but if you can locate a large word list for the language of interest, you can use that list to replace the `AC_Words.txt` file provided with Boxer.

Set Trigger Key
Use the *Set Trigger Key* button to change the key that is used to complete a partially typed word, or expand a User-Defined phrase with trigger style activation.

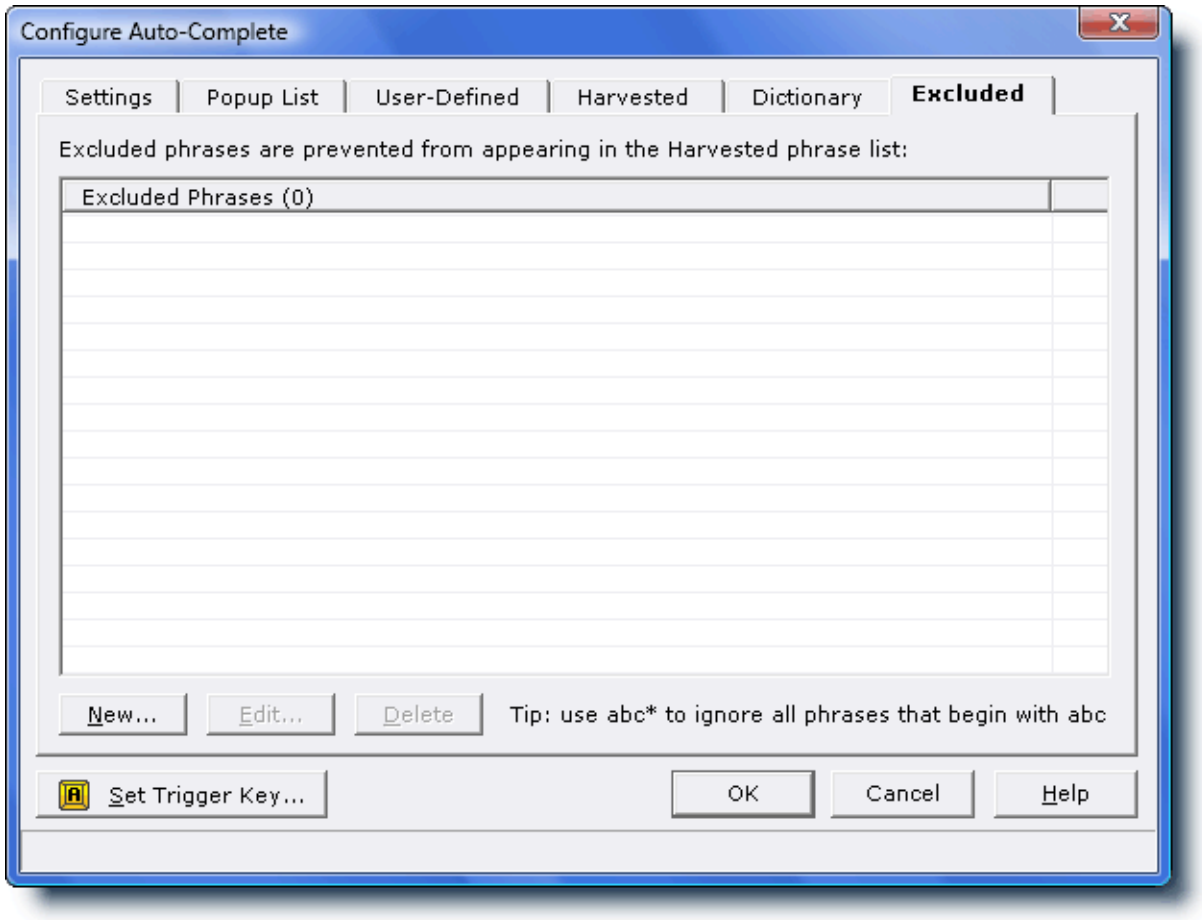
5.22 Auto-Complete - Excluded

Menu: Configure > Auto-Complete

Default Shortcut Key: none

Macro function: none

The Excluded tab of the Configure Auto-Complete dialog can be used to view or edit the Excluded word list. Words that appear in the Excluded word list will never appear in the popup list of matching words, even though they might otherwise be eligible to appear there. If you find that a certain word is being suggested for completion, and you find its presence in the popup list to be bothersome, simply add that word to the Excluded word list.



New
Use the *New* button to add a word to the Excluded word list.

You can use a string of the form `abc*` to cause all phrases beginning with `abc` to be excluded.

Edit
Use the *Edit* button to edit an existing word in the list.

Delete
Use the *Delete* button to delete a word from the list.

Set Trigger Key
Use the *Set Trigger Key* button to change the key that is used to complete a partially typed word, or expand a User-Defined phrase with trigger style activation.

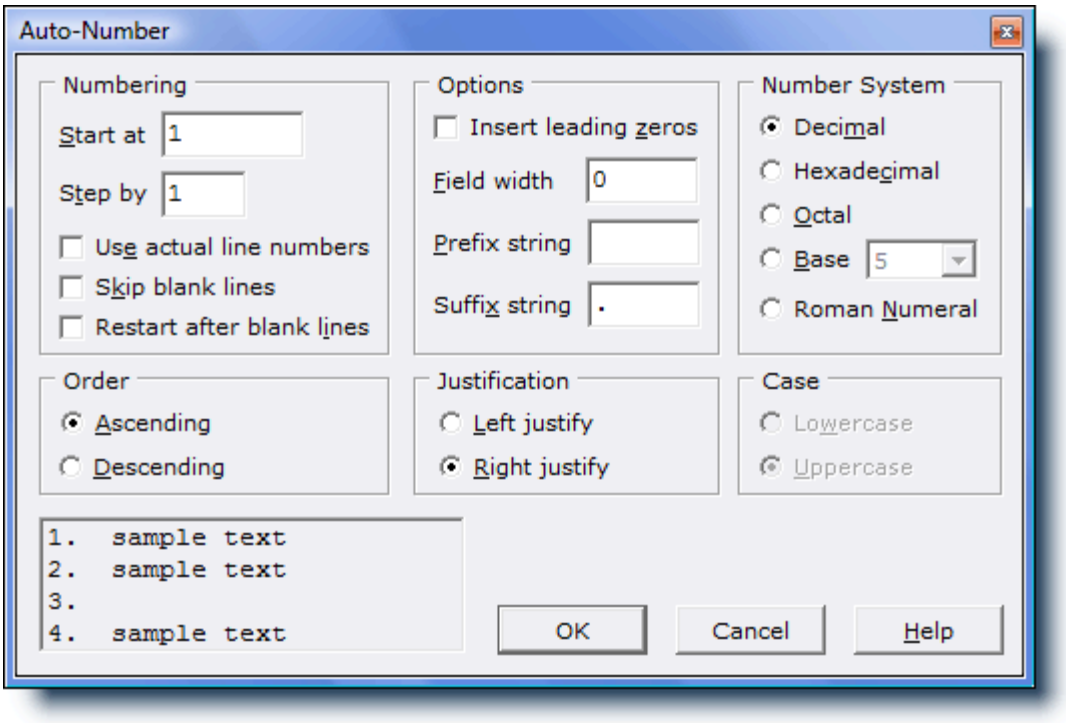
5.23 Auto-Number

Menu: Block > Auto-Number

Default Shortcut Key: None

Macro function: AutoNumber()

The Auto-Number command can be used to automatically number a selected range of lines. A variety of options are available to control the numbering operation, and are described below:



Numbering

Start at
This is the value that will be used to start the numbering from.

Step by
This is the increment that numbering will jump by from line to line. Programmers might use 10 or 100 for program listings, for example.

Use actual line numbers
This option allows the line number for each line to be used. When this option is used, the *Start at* and *Step by* options are disabled.

Skip blank lines
Use this option to control whether or not blank lines will be numbered.

Restart after blank lines
This option causes line numbering to restart from the starting value after a sequence of one or more blank lines is encountered.

Order

Ascending
Use this option for numbering which increases in value.

Descending
Use this option for numbering which decreases in value.

Options

Insert leading zeros
This option can be used to force leading zeros on the numbers.

Field width
Use the field width property to control the width of the numbers that will be generated. Use '0' for automatic sizing.

Prefix string
Use this edit box to specify the text to be placed at the left of the numbers.

Suffix string
Use this edit box to specify the text to be placed at the right of the numbers.

Justification

Left justify
This option can be used for numbering which is left aligned.

Right justify
This option can be used for numbering which is right aligned.

Number System

Decimal

Use this option for numbering in the *decimal* system.

Hexadecimal
Use this option for numbering in the *hexadecimal* system. The case of the alphabetic characters used can be controlled with the *Lowercase* and *Uppercase* options below.

Octal
Use this option for numbering in the *octal* system.

Base
Use this combobox to select any base in the range 2 to 36. For bases 11 and above, alphabetic characters are used in place of digits. The case of the alphabetic characters used can be controlled with the *Lowercase* and *Uppercase* options below.

Roman Numeral
Use this option for numbering with Roman Numerals. The case of the alphabetic characters used can be controlled with the *Lowercase* and *Uppercase* options below.

Case

Lowercase
This option can be used to dictate that lowercase characters be used when *Hexadecimal* or *Roman Numeral* numbering is in use.

Uppercase
This option can be used to dictate that uppercase characters be used when *Hexadecimal* or *Roman Numeral* numbering is in use.

5.24 Auto-Update

Menu: Project > Auto-Update

Default Shortcut Key: none

Macro function: ProjectAutoUpdatel()

This option can be used to keep the editing options of the active project updated automatically. As windows sizes, window locations, bookmarks, cursor locations and other file-specific options change, the project file will be updated automatically. Each project is permitted to have a different Auto-Update setting, if desired.

If you prefer to maintain project settings manually, the Update One and/or Update All commands can be used to manually update the editing options for the current file, or all open files, on an as-needed basis.

5.25 Backtab

Menu: Jump > Backtab

Default Shortcut Key: Alt+Left Arrow

Macro function: Backtab()

The Backtab command is used to move the text cursor backward to the previous tabstop. The size of tabstops is determined by the Tab Display Size command.

5.26 Bookmark Manager

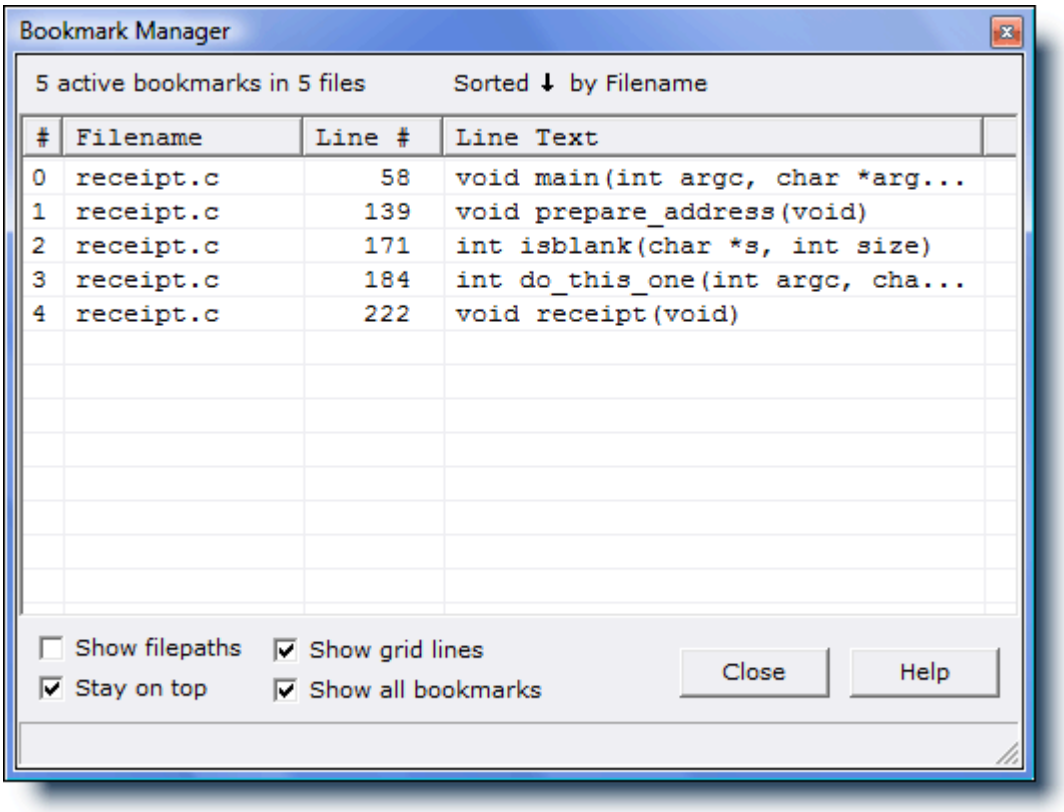
Menu: Jump > Bookmark Manager

Default Shortcut Key: Shift+F9

Macro function: BookmarkManager()

The Bookmark Manager command displays a pop-up dialog showing all bookmarked lines in the files being edited. Double clicking on an entry causes the associated file to become current, and the cursor to be placed on the bookmarked line. The display can be sorted on any of the fields by clicking on the header bar at the top of each field. Press the *Delete* key to remove a bookmark.

The Bookmark Manager can be left open while working in Boxer, so that it's available for reference or quick navigation.



Show filepaths
Use this option to control whether filenames or full filepaths are displayed for each bookmark entry.

Stay on Top
This checkbox controls whether or not the dialog will remain on top of other windows.

Show Grid Lines
Use this option to toggle on/off the display of grid lines within the view.

Show all bookmarks
This checkbox can be used to control whether bookmarks are displayed for all open files, or for only the current file.

Bookmarks will persist for the current editing session, and will be restored when restoring an edit session.

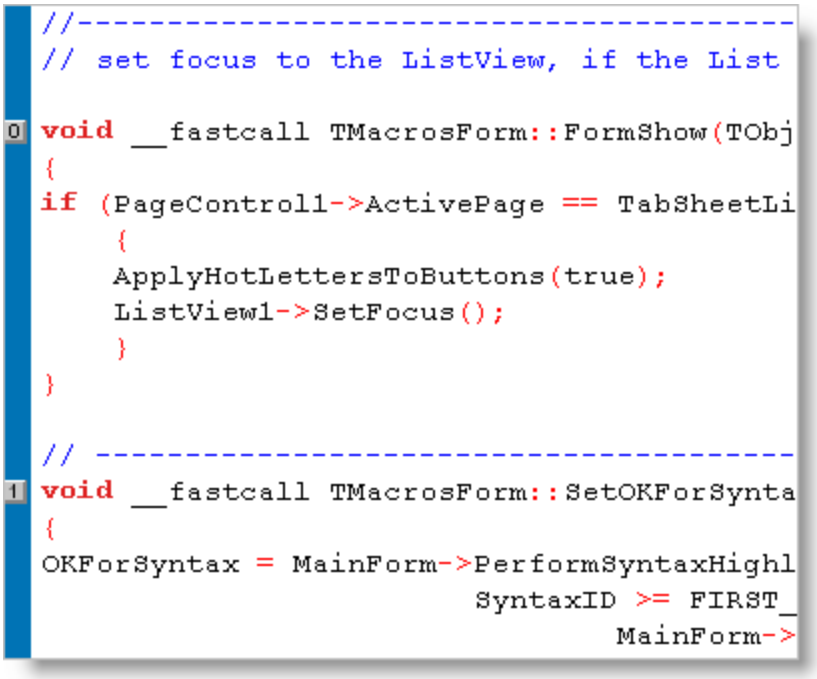
5.27 Bookmarks

Menu: View > Bookmarks

Default Shortcut Key: Alt+F2

Macro function: ViewBookmarks()

The View Bookmarks command is used to toggle on or off the bookmarks in the left column of the editor window. When active, bookmarked lines are displayed with a small number (0-9) in a region to the left of the editing space:



```
//---------------------------------------
// set focus to the ListView, if the List

void __fastcall TMacrosForm::FormShow(TObj
{
if (PageControl1->ActivePage == TabSheetLi
    {
    ApplyHotLettersToButtons(true);
    ListView1->SetFocus();
    }
}

// ---------------------------------------
void __fastcall TMacrosForm::SetOKForSynta
{
OKForSyntax = MainForm->PerformSyntaxHighl
                        SyntaxID >= FIRST_
                                MainForm->
```

The Toggle Bookmark command is used to set or clear a bookmark on the current line at the current column of the text cursor. The Previous Bookmark and Next Bookmark commands can be used to move among bookmarked lines.

Whether or not bookmarks are displayed, they remain functional. All bookmark commands are available even when View Bookmarks is toggled off.

Clicking in the Bookmark region with the right mouse button provides access to its context menu, which allows the display of Bookmarks to be turned off.

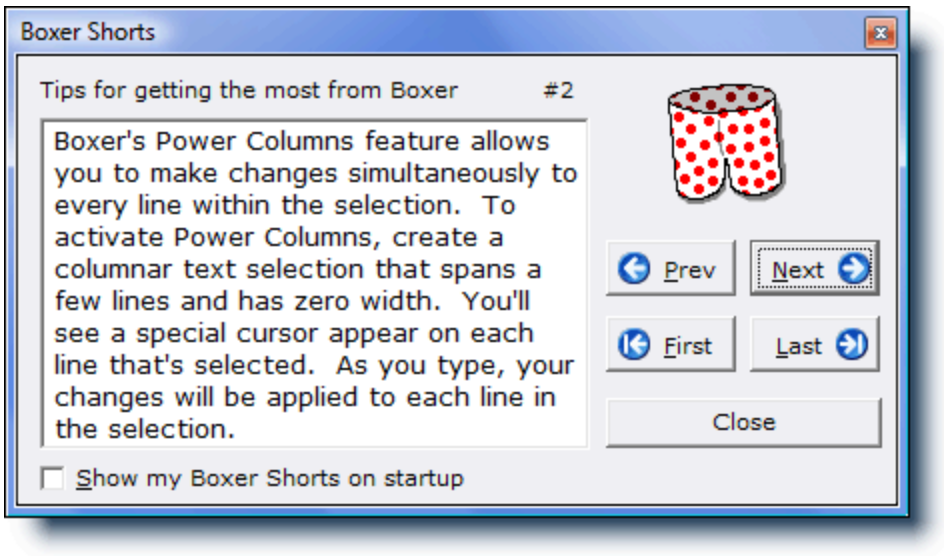
The background color of the bookmark region is shared with that of the Line Numbers. Use the Configure Colors command to select screen colors.

5.28 Boxer Shorts

Menu: Help > Boxer Shorts

Default Shortcut Key: none

'Boxer Shorts' are a collection of useful tips which are displayed in a popup window:



The tips presented in Boxer Shorts will help you to discover some of Boxer's lesser known features and capabilities. You can page through the tips with the buttons supplied, or select the *Remember to 'pull up' your Boxer Shorts upon startup* option so that the tips will be shown each time Boxer starts.

If you think you have a clever tip which could benefit other Boxer users, please send it to info@boxersoftware.com. We'll include the best tips we receive in future editions of Boxer Shorts.

5.29 Boxer Software Order Form

If you prefer not to order online, use the *Print* button to print this order form, and then mail or fax it to us.

BOXER SOFTWARE ORDER FORM

NAME: _____

(COMPANY:) _____

ADDRESS: _____ CITY/TOWN: _____

STATE/PROV: _____ ZIP/POST CODE: _____

COUNTRY: _____ PHONE: _____

EMAIL: _____

MC/VISA/DISCOVER/AMEX: _____ Exp ___ /___

CARDHOLDER: _____

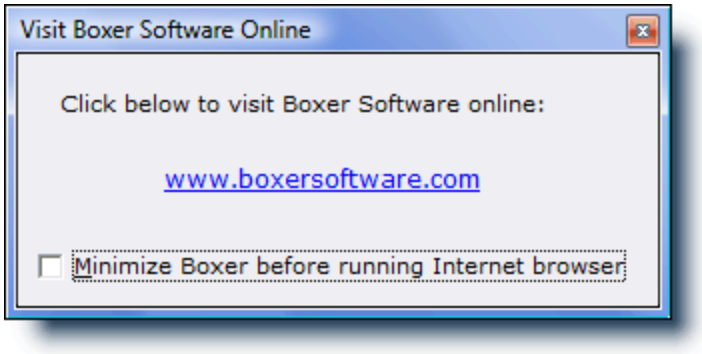
```
 SIGNATURE:  _____ DATE: ____ /____ /____

HOW DID YOU FIRST LEARN OF BOXER SOFTWARE? _____

FOR WHAT TASKS DO YOU USE OUR PRODUCT?  _____

QTY  DESCRIPTION                                  PRICE   TOTAL

___  Boxer Text Editor                            $59.99 ____.__

___  Text Monkey                                   29.99 ____.__

___  The Permutator                                49.99 ____.__

___  File Append and Split Tool                    19.95 ____.__

___  Shipping: $4.00 U.S./Canada, $6.00 elsewhere       ____.__

                                            TOTAL:       ____.__

_____________________________________________________________________

  Boxer Software, Post Office Box 14545, Scottsdale, AZ 85267-4545
Sales: 800-982-6937   Voice: +1-602-485-1635  Fax  :
+1-602-485-1636
      sales@boxersoftware.com         www.boxersoftware.com
```

5.30 Boxer Software Website

Menu: Help > Boxer Software Website

Default Shortcut Key: none

This command displays a dialog box with the address of the Boxer Software Website. Just click the address and your Internet browser will be run, taking you to our website. If you prefer that Boxer be minimized before the browser is run, an option is provided to do so.



The Boxer Software Website has a host of information that will be of interest to all users of Boxer. We invite you to visit the site periodically to learn about new products, upgrades, and usage tips, and for other information which will be posted.

You can also visit our site by clicking here: www.boxersoftware.com

5.31 Bring User Lists to Top

Menu: Tools > User Lists > Bring User Lists to Top

Default Shortcut Key: none

Macro function: BringUserListsToTop()

Use this command to bring all open User List windows to the top of the desktop.

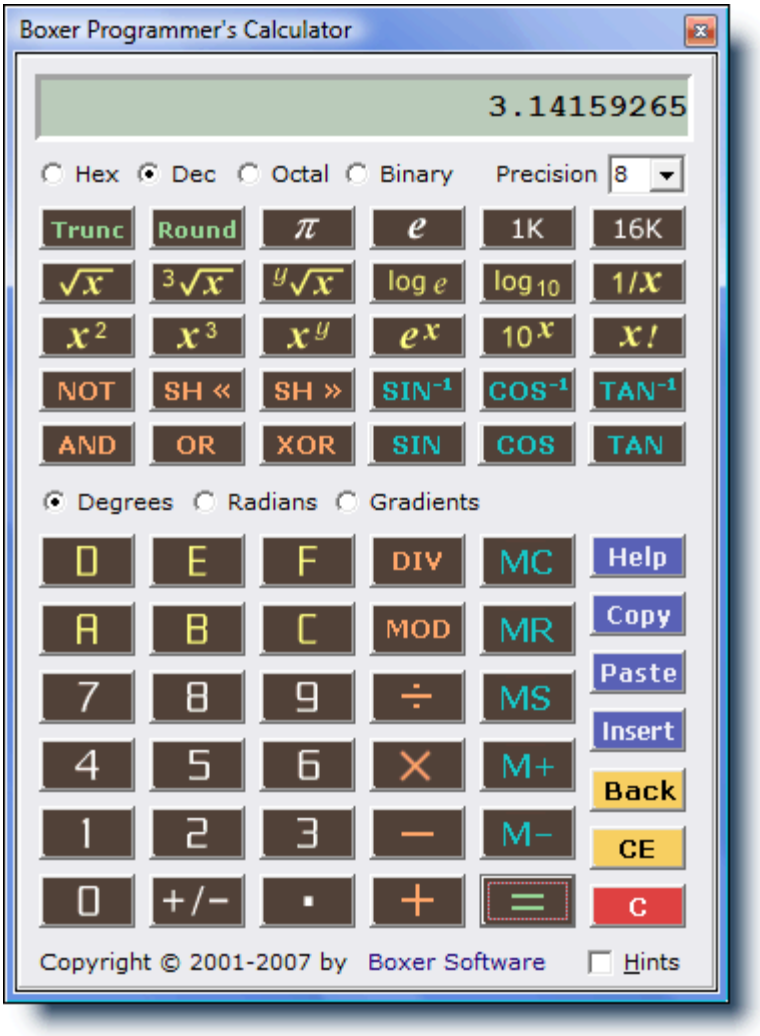
5.32 Calculator

Menu: Tools > Calculator

Default Shortcut Key: F11

Macro function: Calculator()

The Calculator command provides access to Boxer's multi-function, multi-base calculator:



Boxer's Calculator works just like a conventional calculator, and it has all the scientific and trigonometric functions one would expect to find on a full-featured calculator. Values can be entered by clicking keys with the mouse, or by using the keyboard. A list of shortcut keys can be found below.

If a numeric value appears beneath the text cursor when the Calculator is summoned, that value will be placed into the calculator display automatically. The Calculator is also able to interact with the clipboard. The *Copy* button can be used to copy the value in the Calculator's display to the Windows clipboard. The *Paste* button can be used to paste a value from the clipboard into the Calculator's display. The *Insert* button will insert the value in Calculator display at the current text cursor location.

Selecting the *Hints* checkbox reveals a small panel at the bottom of the Calculator that displays information about the key below the mouse cursor.

Calculator Shortcut Keys

Degrees Mode	F2
Radians Mode	F3

Grads Mode	F4
Hexadecimal Mode	F5
Decimal Mode	F6
Octal Mode	F7
Binary Mode	F8
Precision	F10
Pi	P
Euler's Constant	E
1 Kilobyte	K
16 Kilobytes	Ctrl+K
Square Root	Q
Log - natural	N
Log - base 10	L
Reciprocal	R
Square	@
Cube	#
Y-th Power	Y
e to the X	X
Factorial	!
NOT	~
AND	&
OR	|
XOR	^
Shift Left	<
Shift Right	>
Sine	S
Cosine	O
Tangent	T
Add	+
Subtract	-
Multiply	*
Divide	/
Modulus	%
Memory Add	Ctrl+P
Memory Subtract	Ctrl+S
Memory Store	Ctrl+M
Memory Recall	Ctrl+R
Memory Clear	Ctrl+L
Copy to Clipboard	Ctrl+C
Paste to Display	Ctrl+V
Insert into File	Ctrl+I
Clear	Esc
Clear	C (unless in hex mode)
Clear Entry	Del
Back	Backspace
Help	F1
Close	Alt+F4

☞ The Calculator uses 64-bit arithmetic so that *very large* values can be entered and computed. Special thanks are due to Jonas Hammarberg for his help in this area.

☞ The Calculator uses the Algebraic Operating System (AOS), not Reverse Polish Notation (RPN).

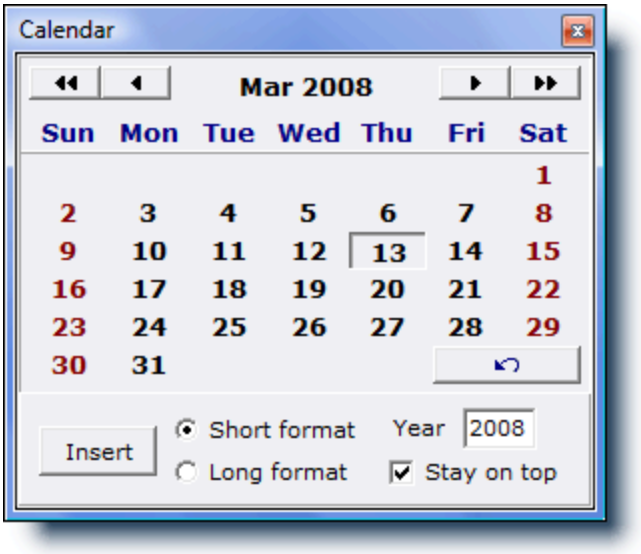
5.33 Calendar

Menu: Tools > Calendar

Default Shortcut Key: none

Macro function: Calendar()

The Calendar command provides access to Boxer's popup calendar:



When first summoned, the Calendar displays the current month and year and highlights the current date. The arrow buttons at the top of the display can be used to move forward or backward, by months or by years. The button with the curved arrow can be used to return the display to the current month and year.

Clicking on any date within the display highlights that date. The *Insert* button can be used to insert a text string describing the highlighted date at the text cursor location. The *Short Format* and *Long Format* options control the format that will be used. The short and long formats used to display the date are in accordance with the regional settings for date display as defined on your computer. To change these settings, see Start Menu | Settings | Control Panel | Regional Settings | Date.

The Calendar recognizes various characters to speed movement from date to date:

```
Y = first day of year
R = last  day of year
M = first day of month
H = last  day of month
T = Today
```

The following keys are also recognized:

```
Space         = +1 month
-             = -1 day
+             = +1 day
Shift+Left    = -1 month
Shift+Right   = +1 month
Ctrl+Left     = -1 year
Ctrl+Right    = +1 year
```

The day on which a calendar week starts can be configured on the Configure | Preferences | Other options page. The option is titled: *Calendar week starts on*.

If you prefer that the Calendar always remain on top of other windows, the *Stay on top* option can be used. The Calendar is a *non-modal* window, which allows it to remain on-screen after *focus* has been returned to another editing window.


If the Calendar is left on-screen when Boxer is closed, it will be automatically reopened if the edit session is later restored.

5.34 Cascade

Menu: Window > Cascade

Default Shortcut Key: none

Macro function: Cascade()

The Cascade command can be used to resize and reposition all editor windows in a cascading arrangement beginning at the upper left of the *client area* and proceeding to the lower right. The height and width of each window is uniform, and is determined by the size of the client area.

Minimized windows are not affected by this operation.

5.35 Cascade Vertical

Menu: Window > Cascade Vertical

Default Shortcut Key: none

Macro function: CascadeVertical()

The Cascade Vertical command can be used to resize and reposition all editor windows in a cascading arrangement beginning at the upper left of the *client area* and proceeding toward the lower left. The height and width of each window is uniform, and is determined by the size of the client area. Unlike the Cascade command, the left edges of all windows are placed against the left edge of the client area.

Minimized windows are not affected by this operation.

5.36 Cascade Horizontal

Menu: Window > Cascade Horizontal

Default Shortcut Key: none

Macro function: CascadeHorizontal()

The Cascade Horizontal command can be used to resize and reposition all editor windows in a cascading arrangement beginning at the upper left of the *client area* and proceeding toward the upper right. The height and width of each window is uniform, and is determined by the size of the client area. Unlike the Cascade command, the top edges of all windows are placed against the top edge of the client area.

Minimized windows are not affected by this operation.

5.37 Check for Latest Version

Menu: Help > Check for Latest Version

Default Shortcut Key: none

This command can be used to check if the version of Boxer being used is still up-to-date. This command will launch your Internet browser to display a special page at the Boxer Software website. If a new version of Boxer is available, details will be given on how to get the update. Minor updates to Boxer will be made available free-of-charge. See the topic Upgrade Information for more information.

In order to launch your Internet browser, Boxer relies upon the operating system shell's ability to open an Internet address. When an Internet browser is installed, it typically establishes itself as the program which is called by the shell to open such addresses. This is true of all common browsers you are likely to encounter. If you find that your Internet browser is *not* launched by Boxer, or if some other inactive browser is launched instead, it's because your active browser has not established itself as the one that processes the 'open' request from the operating system shell for Internet addresses. This situation should be rare, cannot be remedied by Boxer, and is not due to any shortcomings in Boxer.

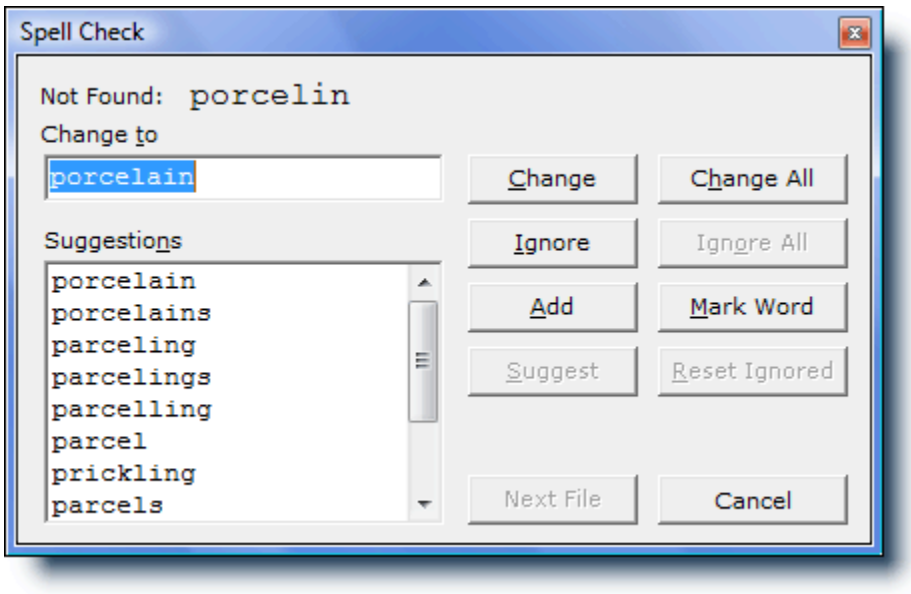
5.38 Check Word

Menu: Tools > Check Word

Default Shortcut Key: Shift+F7

Macro function: CheckWord()

Use the Check Word command to check the spelling of the word at the text cursor. If the word is spelled correctly, a message will appear to confirm this fact. If the word is spelled incorrectly, a dialog will appear providing options to make a correction:



The Check Word command can also be accessed by right-clicking on a suspect word and selecting the command from the context menu.

5.39 Clear All Clipboards

Menu: Edit > Clear Clipboard > All Clipboards

Default Shortcut Key: none

Macro function: ClearAllClipboards()

The Clear All Clipboards command can be used to clear (erase) the content of all clipboards. The Windows clipboard and the eight internal clipboards will all be affected.

The effect of the Clear All Clipboards command cannot be undone with Undo, so use this command carefully.

5.40 Clear Clipboard

Menu: Edit > Clear Clipboard > Clipboard *n*

Default Shortcut Key: none

Macro function: ClearClipboard()

The Clear Clipboard command can be used to clear (erase) the content of the clipboard selected. The content of each clipboard is displayed in a popup window as the menu cursor is moved across the clipboard's menu entry. This makes it easy to check what's on a clipboard before deciding whether to clear it.

The effect of the Clear Clipboard command cannot be undone with Undo, so use this command carefully.


When the content of a clipboard is displayed in a popup window, the text is displayed with an 8 point, fixed width, *Courier New* font. This font utilizes the ANSI character set mapping. If the current screen font uses an OEM character set mapping, and if characters outside the normal alphanumeric range reside on the clipboard, then the content of the clipboard may appear different in the popup window than it would in the underlying file. This difference is simply the result of a difference in character sets, and does not mean that the data on the clipboard has been adjusted or corrupted.


The Clear Windows Clipboard command will remain enabled even when the Windows Clipboard contains non-text data. This allows the content of the clipboard to be cleared by Boxer, in case this operation is desired.

5.41 Clear Closed Tabs List

Menu: File Tab Context Menu > Closed Tabs List > Clear List

Default Shortcut Key: none

Macro function: ClearClosedTabsList()

Clear the list of closed file tabs

5.42 Clear Recent Files List

Menu: File > Clear Recent Files List

Default Shortcut Key: None

Macro function: ClearRecentFilesList()

Use this command to clear the record of recently accessed files from the Recent Files

submenu.

5.43 Clear Recent Projects List

Menu: Project > Clear Recent Projects List

Default Shortcut Key: none

Macro function: ClearRecentProjectsList()

Use this command to clear the record of recently accessed projects from the Recent Project submenu.

☞ See the Project | New command for full details about Boxer's project file feature.

5.44 Clear Undo

Menu: Edit > Clear Undo

Default Shortcut Key: none

Macro function: ClearUndo()

The Clear Undo command can be used to clear the record of changes which are used by the Undo and Redo commands. After issuing this command, the record of changes for the current file is erased, and the Undo and Redo commands become disabled until additional changes are made.

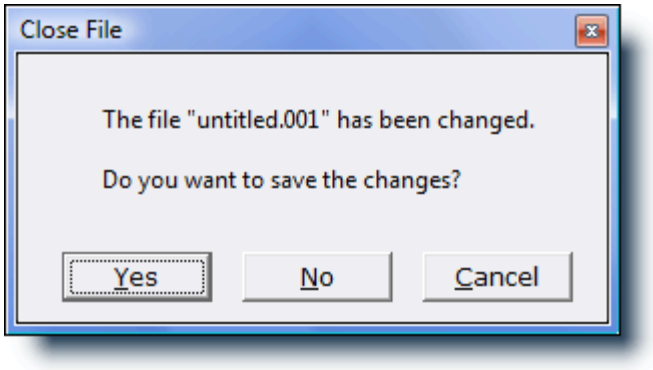
5.45 Close (File)

Menu: File > Close

Default Shortcut Key: Alt+X

Macro function: Close()

The Close command is used to close the file in the active editor window. If unsaved changes have been made to the file, a dialog box will first appear to alert you to this fact. You will then be able to choose whether to save the changes before closing, close without saving, or cancel the Close operation altogether.



You can quickly tell whether a file has unsaved changes by looking for an asterisk (*) to the left of its name in the title bar, or on its File Tab.

A file can also be closed by clicking the 'X' box in the upper right corner of its window. When a file's window is maximized, the 'X' box will be located at the far right of the main menu bar.

If you would prefer that Boxer be minimized automatically when the last file is closed, there is a checkbox on the Configure | Preferences | Other options page to achieve this. The option is titled *Minimize Boxer when closing last file.*

5.46 Close (Project)

Menu: Project > Close

Default Shortcut Key: none

Macro function: ProjectClose()

Use the Project | Close command to close the current project. All files associated with the active project will be closed.

☞ See the Project | New command for full details about Boxer's project file feature.

5.47 Close All

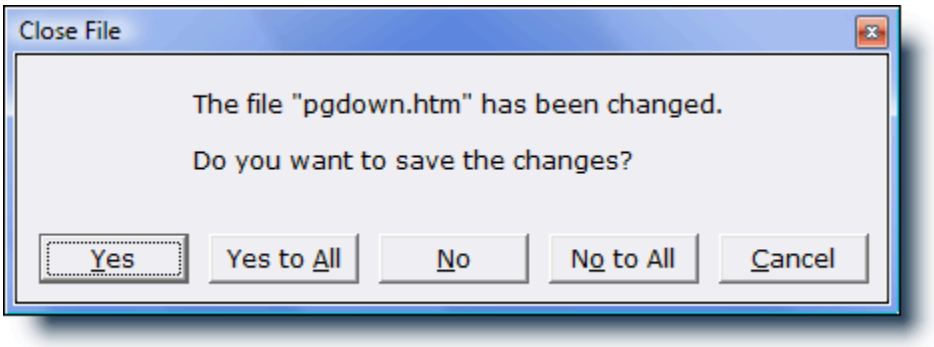
Menu: File > Close All

Default Shortcut Key: none

Macro function: CloseAll()

The Close All command is used to close all open files within the editor. If unsaved changes have been made to any file, a dialog box will appear for each such file to alert you to this fact. You will then be able to choose whether to save the changes before closing, close without saving, or to cancel the Close All operation. If Close All is issued when more than one file is modified, *Yes-to-All* and *No-to-All* buttons are provided to

save time:



You can tell quickly whether a file has unsaved changes by looking for an asterisk (*) to the left of its name in the title bar, or on its File Tab.

If you would prefer that Boxer be minimized automatically when the last file is closed, there is a checkbox on the Configure | Preferences | Other options page to achieve this. The option is titled *Minimize Boxer when closing last file*.

The File | Close All command is functionally equivalent to the Window | Close All command, since each file resides in its own window.

5.48 Close All but Active

Menu: Window > Close All but Active

Default Shortcut Key: none

Macro function: CloseAllButActive()

Use this command to close all open editing windows *except* the current window.

You can quickly tell whether a file has unsaved changes by looking for an asterisk (*) to the left of its name in the title bar or on its File Tab.

5.49 Closed Tabs List

Menu: View > File Tabs > Undo Close Tab

Default Shortcut Key: none

Macro function: UndoCloseTab()

The Undo Close Tab command can be used to reopen the file that was last closed during the current editing session. This command makes it easy to reopen a file if it was closed accidentally.

The "Closed Tabs List" at the bottom of the View | File Tabs submenu shows the names of the files that are eligible to be reopened, and allows files within the list to be selectively reopened.

Clicking the middle mouse button in an open area of the file tab bar is taken as a shortcut gesture to reopen the last closed file tab.

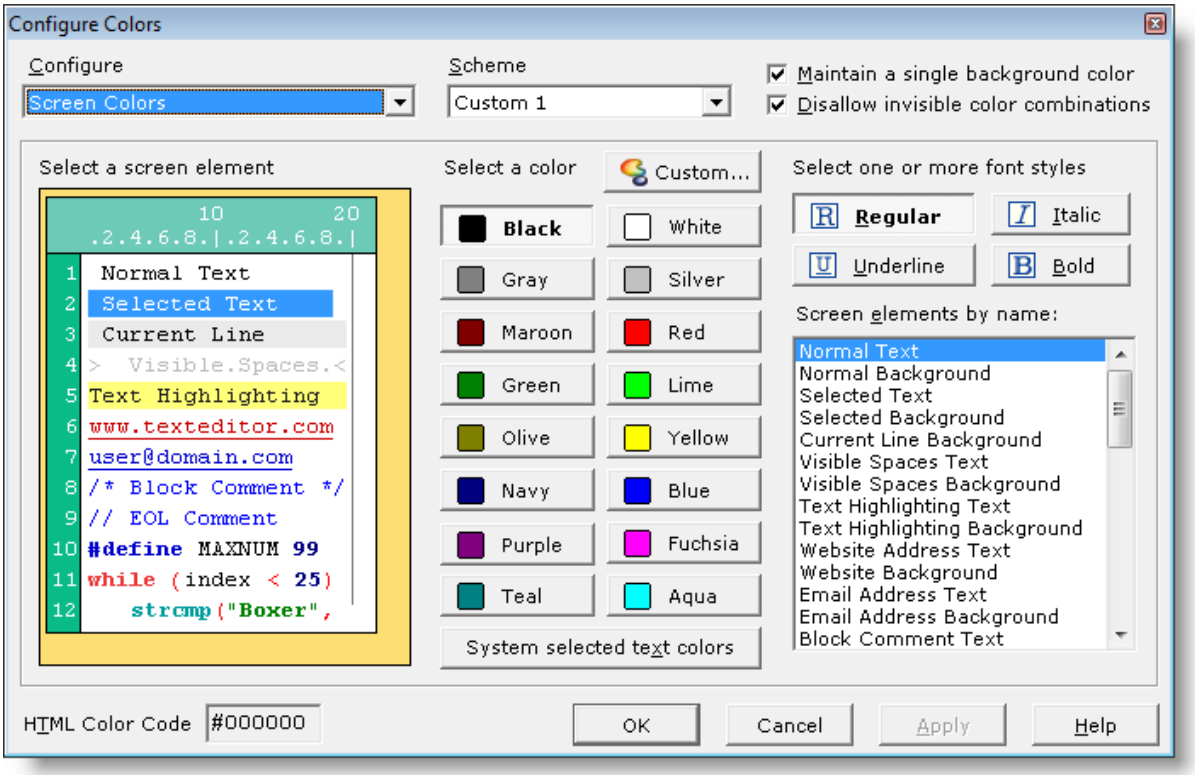
5.50 Colors

Menu: Configure > Colors

Default Shortcut Key: none

Macro function: ConfigureColors()

The Configure Colors command allows you to customize the colors that are used to display and print text files. Buttons are provided for the standard 16 colors. Use the *Custom* button to select from a palette of 16 million colors.



Configure
The Configure Colors dialog box operates in three different modes: *Screen Colors*, *Color Syntax Printing*, and *Monochrome Syntax Printing*. The active mode is selected from the *Configure* drop-down list at the upper left of the dialog box. *Screen Colors* mode allows you to set the colors used to display text files on-screen. *Color Syntax Printing*

mode is used to set the colors used for printing program files on color printers. *Monochrome Syntax Printing* mode is used to set the colors used for printing program files on non-color printers.

Scheme
A set of pre-defined color schemes has been provided to speed the process of color configuration. You can start with the pre-defined scheme that is closest to your liking, and then make other changes as desired. Once a change has been made to a pre-defined color scheme you will be asked where to save the custom layout; four custom positions are available.

Maintain a single background color
This option can be used to ensure that all elements will use a single background color. When selected, Boxer ensures that all elements are updated when the background color is changed. Turn this option off if you would like to create a color scheme in which some elements use different background colors.

Disallow invisible color combinations
This option can be used to prevent any color selection which would cause the foreground and background colors of one or more elements to be the same.

System selected text colors
This button can be used to quickly apply the default text selection colors of the operating system to the current color scheme.


This option will be of particular use to blind users who are using Boxer with a screen reader such as JAWS. Screen reader software sometimes requires that the text selection colors used by an application match those that are used system wide.

HTML Color Code
As a convenience, the current color is displayed in HTML Color Code format to make it easy to duplicate a color used in Boxer in your HTML code. The HTML Color Chart command can also be used for selecting colors and getting an HTML Color Code value.

The process of changing colors is quite simple, and includes three steps:

1. Click to select an element
Click with the left mouse button in the miniature screen display on the element which is to be changed. You can click on the text of an element to select its foreground element, or on the background of an element to select its background element. After clicking, you'll see that the *Elements by name* listbox is updated to reflect the selected element. You'll also see that the color and font style buttons will be displayed in a depressed state to reflect the current settings for the selected element. If the element uses a color other than those appearing on the standard buttons, the *Custom* button will appear depressed. Some elements, such as the Right Margin Vertical Rule, cannot be easily selected by mouse in the miniature screen display. These elements can be selected from the *Elements by name* listbox instead.

2. Click to select a color
Click on the new color for the selected element. The miniature screen display will be

updated to reflect the new color. When configuring for Monochrome Syntax Printing, the available colors will be reduced to those which can be achieved on non-color printers.

3. Click to select font style(s)
Click on one or more font styles for the selected element. The miniature screen display will be updated to reflect the new style. To remove a font style, click again on its button to clear the style.


When configuring screen colors, the *Apply* button can be used to update the screen below the dialog box to reflect the changes made.

5.51 Command Multiplier

Menu: Tools > Command Multiplier

Default Shortcut Key: Alt+Y

Macro function: none (Boxer's macro language provides far more powerful methods to multiply the execution of commands)

The Command Multiplier can be used to multiply the execution of a keystroke or command key sequence. A popup box appears to retrieve the multiplier to be used. After clicking OK, Boxer awaits the next keystroke or command key sequence. Once issued it will be performed repeatedly, according to the value entered.

This command might be used to multiply the execution of an insertable character so as to create a divider bar containing a known number of characters. Or it might be used with the Delete Current Line command to quickly delete 100 lines.


To simply repeat the last command issued, one or more times, use the Repeat Last Command command.

5.52 Comment

Menu: Block > Comment

Default Shortcut Key: F5

Macro function: Comment()

The Comment command can be used to apply commenting to the current line--or to a selected range of lines--when editing a file for which Boxer has syntax information defined (see Configure | Syntax Highlighting).

When text is not selected, the current line will be commented using the end-of-line comment sequence for the language being edited. If that sequence is not available, the entire line will be enclosed using the open and close comment sequences. In either case the text cursor is advanced to the line below following the operation.

If text is selected, the selected lines will be bracketed with the open and close block comment sequences for the language being edited. If the language does not support block commenting, the end-of-line comment sequence will be applied to each line within the selected range. If neither of these sequences has been defined, an error message will be given.

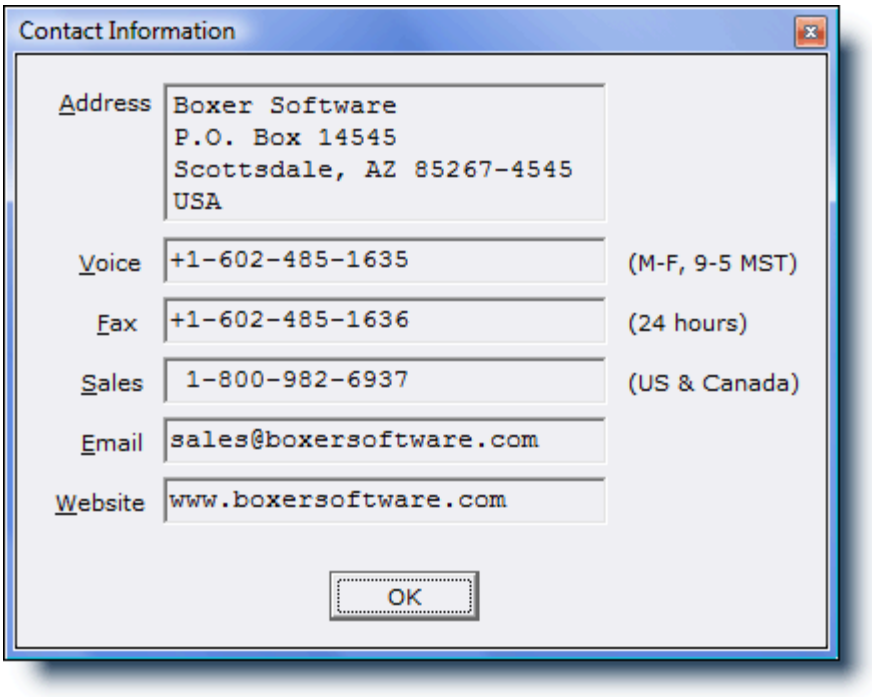
The Uncomment command can be used to remove commenting from the current line or from selected text.

5.53 Contact Information

Menu: Help > Contact Information

Default Shortcut Key: none

The Contact Information command displays this dialog:



5.54 Convert Case - Invert

Menu: Block > Convert Case > Invert

Default Shortcut Key: none

Macro function: CaseInvert()

The Invert command converts alphabetic characters within the selected text to the opposite case. Uppercase characters are converted to lowercase; lowercase characters are converted to uppercase.

```
Before conversion: The quick brown fox jumped over the lazy dog.
After conversion : tHE QUICK BROWN FOX JUMPED OVER THE LAZY DOG.
```

Boxer consults the case conversion map provided by the operating system so that accented characters can be properly converted.

5.55 Convert Case - Lower

Menu: Block > Convert Case > Lower

Default Shortcut Key: none

Macro function: CaseLower()

The Lower command converts alphabetic characters within the selected text to lowercase.

```
Before conversion: The quick brown fox jumped over the lazy dog.
After conversion : the quick brown fox jumped over the lazy dog.
```

Boxer consults the case conversion map provided by the operating system so that accented characters can be properly converted.

5.56 Convert Case - Sentences

Menu: Block > Convert Case > Sentences

Default Shortcut Key: none

Macro function: CaseSentences()

The Sentences command converts the first character of all sentences within the selected text to uppercase. For purposes of this command, a sentence is considered to be a series of words that ends with either a period, a question mark, or an exclamation mark.

```
Before conversion: the quick brown fox jumped over the lazy dog.
After conversion : The quick brown fox jumped over the lazy dog.
```

Boxer consults the case conversion map provided by the operating system so that

accented characters can be properly converted.

☞ By default, this command will first convert the selected text to lowercase before capitalizing each sentence. This ensures that the command operates as expected when processing text in all uppercase. Converting to lowercase may disrupt some all-caps words (such as acronyms and proper nouns) that should have remained in uppercase, so you should review the results for accuracy after applying the conversion. If you prefer that the selected text not be forced to lowercase prior to operation, you can change this behavior on the Configure | Preferences | Editing 2 dialog page.

5.57 Convert Case - Title

Menu: Block > Convert Case > Title

Default Shortcut Key: none

Macro function: CaseTitle()

This command converts the selected text to conform to the rules of title case (aka proper case). Grammar experts do not all agree on the precise rules for title case, but most references use these rules:

1. Always capitalize the first word
2. Always capitalize the last word
3. Capitalize all other words, except articles, prepositions and conjunctions which have fewer than five letters

☞ Because the application of title case presumes a knowledge of the language--capitalization depends on parts of speech--this command is limited to operating on English text.

☞ By default, this command will first convert the selected text to lowercase before capitalizing words. This ensures that the command operates as expected when processing text in all uppercase. Converting to lowercase may disrupt some all-caps words (such as acronyms) that should have remained in uppercase, so you should review the results for accuracy after applying the conversion. If you prefer that the selected text not be forced to lowercase prior to operation, you can change this behavior on the Configure | Preferences | Editing 2 dialog page.

5.58 Convert Case - Upper

Menu: Block > Convert Case > Upper

Default Shortcut Key: none

Macro function: CaseUpper()

The Upper command converts alphabetic characters within the selected text to uppercase.

```
Before conversion: The quick brown fox jumped over the lazy dog.
After conversion : THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG.
```

Boxer consults the case conversion map provided by the operating system so that accented characters can be properly converted.

5.59 Convert Case - Words

Menu: Block > Convert Case > Words

Default Shortcut Key: none

Macro function: CaseWords()

The Words command converts the first character of all words within the selected text to uppercase.

```
Before conversion: The quick brown fox jumped over the lazy dog.
After conversion : The Quick Brown Fox Jumped Over The Lazy Dog.
```

Boxer consults the case conversion map provided by the operating system so that accented characters can be properly converted.


By default, this command will first convert the selected text to lowercase before capitalizing each word. This ensures that the command operates as expected when processing text in all uppercase. Converting to lowercase may disrupt some all-caps words (such as acronyms) that should have remained in uppercase, so you should review the results for accuracy after applying the conversion. If you prefer that the selected text not be forced to lowercase prior to operation, you can change this behavior on the Configure | Preferences | Editing 2 dialog page.

5.60 Copy

Menu: Edit > Copy

Default Shortcut Key: Ctrl+C

Macro function: Copy()

The Copy command copies the selected text in the current file onto the current clipboard. The current clipboard might be the Windows clipboard or one of Boxer's eight internal clipboards. See the Edit | Set Clipboard command for details on changing the current clipboard.

If text is *not* selected, the Copy command will operate on the current line. This behavior can be controlled on the Configure | Preferences | Editing 1 options page. The

option is titled *Cut/Copy/Append commands use current line when no text is selected*.

When a columnar selection (Block | Select Columnar) is placed on the clipboard, any under-length lines within the selected range will be extended with Spaces to match the width of the rectangular text block. This ensures that the block will behave as expected if the Paste command is later used to insert the text at a new location. Likewise, any Tab characters within the selected region will be converted to Spaces before being placed on the clipboard.


Boxer's clipboard commands will sense the type of text data being placed on the Windows clipboard and, when appropriate, use a more descriptive clipboard format to tag that data. For example, when Boxer senses that HTML code is being copied to the clipboard, the text will also be placed in the HTML clipboard format. This enables a conforming program to paste the data more intelligently. Boxer will also tag Rich Text data (RTF) and Comma-Separated Value (CSV).

5.61 Copy Filename

Menu: Edit > Copy Filename

Default Shortcut Key: none

Macro function: CopyFilename()

The Copy Filename command copies the full filepath of the current window to the current clipboard.

To insert the filepath of the current file into the edited text, use the Insert Filename command.


This command can be useful when supplying the name of an edited file to an email program for attachment to a message.

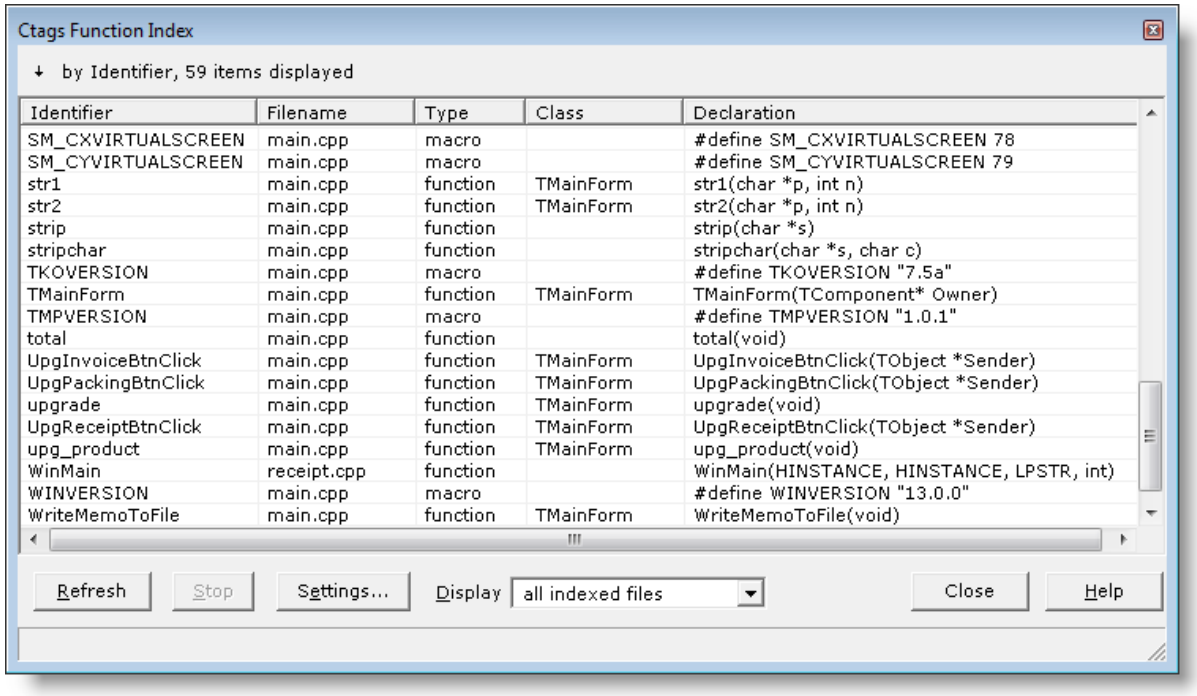
5.62 Ctags Function Index

Menu: Jump > Ctags Function Index

Default Shortcut Key: none

Macro function: CtagsFunctionIndex()

The Ctags Function Index command displays a dialog containing a list of functions, procedures and global variables for the files currently being edited. The list can be used as a handy reference to function names and their calling parameters, or as a navigation aid: double-clicking on an entry will jump to the file and line that corresponds to the highlighted entry. The dialog is *non-modal*, so it can remain open alongside Boxer as you're doing other work.



In order to index the edited files, Boxer runs an external program and then reads the output file it creates. Exuberant Ctags is a fast, multi-language implementation of the original *ctags* and *etags* programs that are available on Unix. Exuberant Ctags is distributed under the *GNU General Public License.* The program `ctags.exe` and a zip file containing the program's source code have been installed in a directory named 'Ctags' beneath the Boxer installation directory.

Exuberant Ctags provides built-in support for indexing the following languages:

Ant	HTML	Ruby
Assembler	Jave	Scheme
ASP	Javascript	Shell scripts (Bourne/Korn/Z)
Awk	Lisp	S-Lang
Basic	Lua	SML (Standard ML)
BETA	Make	Tcl
C and C++	MATLAB	TeX
C#	Objective Caml	Vera
COBOL	Pascal	Veilog
DOS Batch	Perl	VHDL
Eiffel	PHP	Vim
Erlang	PL/SQL	YACC
Flex	Python	

Fortran	REXX

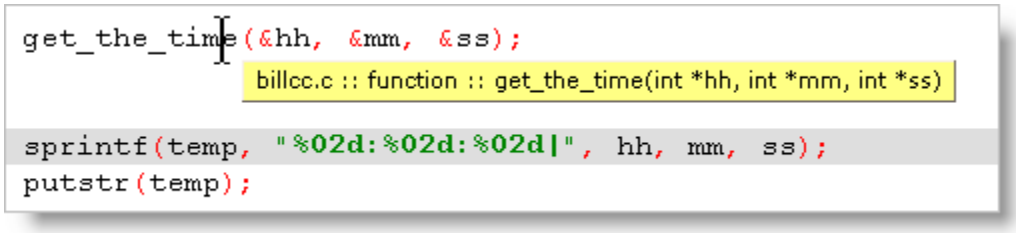
In addition, Boxer is supplied with a `CTAGS.CNF` configuration file that adds support for these languages:

ActionScript	Latex	System Verilog
Cascading Style Sheets	Miva	XML
INI files		

Support for indexing additional languages can be added by making additions to the `CTAGS.CNF` file. The process is not trivial, however, and it's often easier to find a configuration on the internet which has been developed by someone else. Instructions for adding additional languages can be found at the Exuberant Ctags website. By keeping your copy of Ctags up-to-date, you can also be assured of getting access to new built-in languages as they are added by its developers. The version of Ctags that was supplied with Boxer was current at the time of Boxer's release.

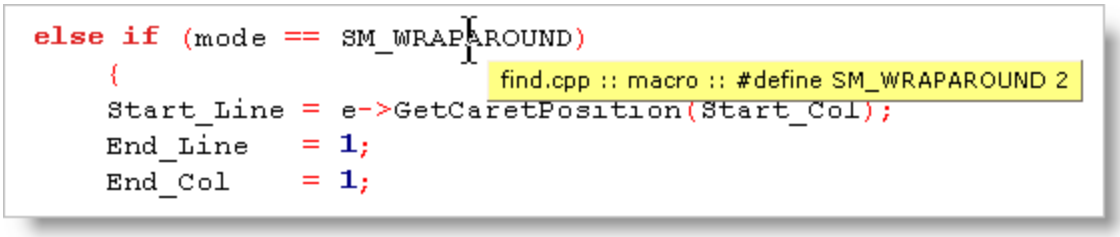
The list can be sorted on any of its columns by clicking on the associated column title in the header at the top of the listing. Clicking on the same header a second time will reverse the order of the sort.

The function prototype information contained in the Ctags Function Index dialog is also available for display when the mouse hovers over a function that has been indexed:



The display of these popup tool tips can be configured by clicking the Settings button, which leads to the Configure Ctags Function Indexing dialog.

Popup tool tips can also be displayed for global variables, structure and class members, typedefs, macros and other language-dependent identifiers:



Refresh
Use the *Refresh* button to re-index all open files, and any other 'extra files' that may have been designated in the Configure Ctags Function Indexing dialog. You might want to use the *Refresh* button when changes have been made to an edited file that would invalidate the information that was previously gathered. For example, if a function's calling parameters are changed, or a function is added or deleted, use *Refresh*.

☞ It is **not** necessary to use *Refresh* simply because the line number of a function's declaration has changed. The indexing is maintained in a format that is not sensitive to changes in line numbers.

Settings
The Settings button will display the Configure Ctags Function Indexing dialog, which provides options that control how and when files will be indexed, the appearance of the function list, and whether popup tool tips will be displayed.

Display
Use the *Display* combobox to filter the listing of indexed functions and variables. The available choices are:

all indexed files
all open files
files in active project
current file
extra files

Extra files to be indexed can be designated in the Configure Ctags Function Indexing dialog.

☞ The *Display* setting will also influence which identifiers are visible to the popup tool tip feature. If you have filtered the listing to show only those entries in the *current file*, for example, you may wish to restore the *Display* setting to *all indexed files* so that the full collection of indexed identifiers are available to the popup tool tips feature.

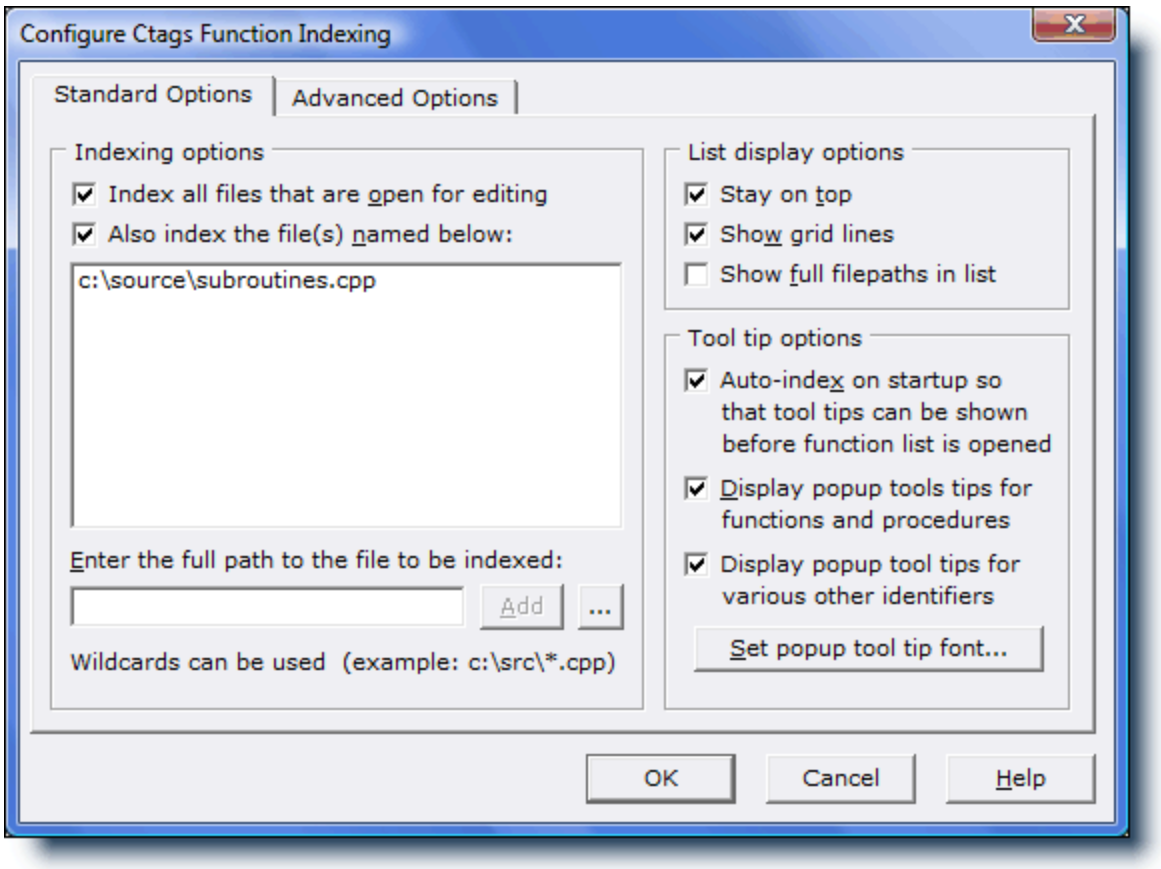
5.63 Ctags Function Indexing

Menu: Configure > Ctags Function Indexing

Default Shortcut Key: none

Macro function: ConfigureCtagsFunctionIndexing()

The Configure Ctags Function Indexing command provides options that relate to the Ctags Function Index feature.



Indexing options

Index all files that are open for editing
When this option is checked, indexing information will be gathered for all files that are open for editing. If the *Auto-index on startup...* option is checked (see below), this indexing will occur automatically shortly after startup, and files that are opened later in the editing session will be indexed as they are opened. If the *Auto-index* option is unchecked, indexing will not occur until and unless the Ctags Function Index command is issued.

☞ When indexing files that are open for editing, please note that the operation is performed on the file as it resides *on disk*, and not on the memory image of the file. If you have made changes to a file that you want to be reflected in the index, be sure to save the file before requesting the indexing operation.

Also index the file(s) named below
When this option is checked, files named in the accompanying list will also be indexed, even if they are not open for editing in the editor. Use this list to name files that you would *always* like to be indexed, even when you're not editing these files. The edit box below the list is used to enter the full filepath of the file to be added. Click the *Add* button to add the file to the list. Use the button with the ellipsis (...) to browse for a file. The *Delete* key can be used to remove an unwanted entry from the list. Right-clicking in the file list will display a context menu, which additionally contains options to edit the selected entry and to delete all entries.

List display options

Stay on top
When checked, this option causes the Ctags Function Index dialog to remain on top of other windows.

Show grid lines
This option controls whether or not grid lines will be displayed between rows and columns in the Ctags Function Index.

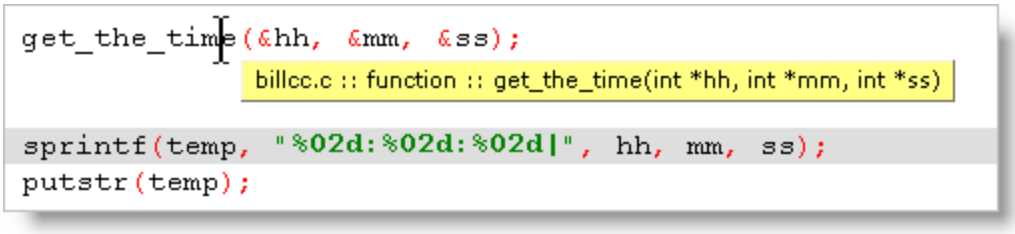
Show full filepaths in list
Use this option if you prefer that full filepaths be displayed in the Ctags Function Index. This option is useful when you're editing files that have the same filename, but reside in different directories.

Tool tip options

Auto-index on startup so that tool tips can be shown before function list is opened
Use this option to ensure that popup function prototype tool tips can be displayed even if the Ctags Function Index command has not been issued.

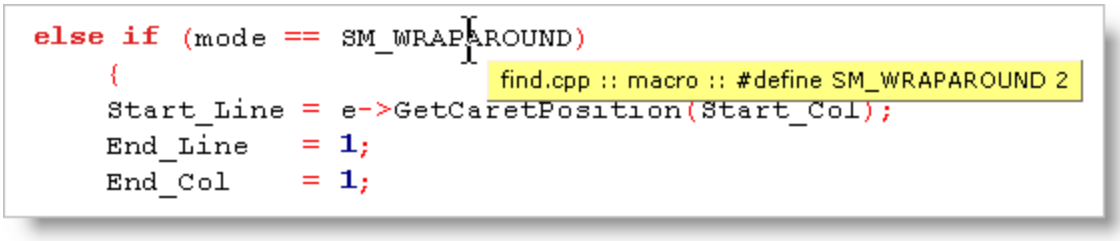

An indexing operation must be performed before function prototypes and global variable information is available for display in either the Ctags Function Index dialog, or in popup tool tips. Depending on the number of files open for editing, the number of *extra files* designated for indexing (see above), the size of these files and the processing speed of your computer, this operation could take anywhere from a split second to several seconds. On modern PC's, and with source files of modest size, the indexing process will be almost instantaneous. However, if you're using a slow PC, or you typically edit many files at once, or your source files are exceptionally large, you may wish to disable auto-indexing. For most situations, the added convenience of having popup information available will outweigh the split-second indexing process.

Display popup tool tips for function and procedures
Use this option to control whether prototypes for functions and procedures will be displayed in popup tool tips. The information that is displayed will be dependent on the language being used. For the C programming language, a popup tool tip for a function might look like this:



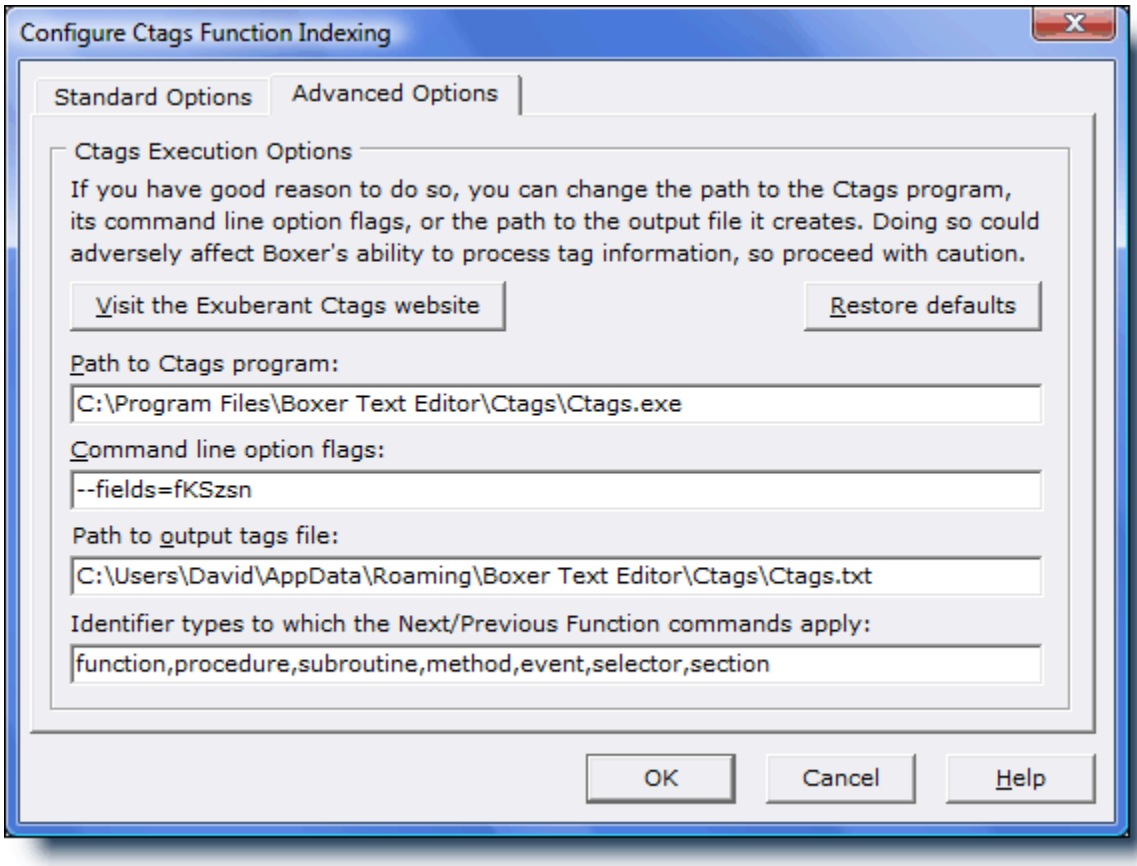
Display popup tool tips for various other identifiers
Use this option to control whether informative tool tips will be displayed for global

variables, structures, members, macros, typedefs and other recognized identifiers. The information that is displayed for an identifier will be dependent on the language being used. For example, when hovering over an identifier that has been `#defined` in the C programming language, the tool tip might look like this:



Ctags Execution Options

If you have good reason to do so, you can change the path to the Ctags program, its command line option flags, or the path to the output file it creates. Doing so could adversely affect Boxer's ability to process tag information, so proceed with caution. You can use the *Restore defaults* button to restore the settings to their recommended values.



Exuberant Ctags supports a *comprehensive* set of command line option flags. With

some experimentation, they can even be used to add support for indexing languages not supported by the program in its as-released form. Full information about the Ctags program can be found at the Exuberant Ctags website.

When run from removable media, Boxer will automatically recompute the "Path to Ctags program" and "Path to output tags file" parameters in case the drive letter of the removable device changed since the last run.

5.64 Cut

Menu: Edit > Cut

Default Shortcut Key: Ctrl+X

Macro function: Cut()

The Cut command removes the selected text from the current file and places it on the current clipboard. The current clipboard might be the Windows clipboard or one of Boxer's eight internal clipboards. See the Edit | Set Clipboard command for details on changing the current clipboard.

If text is *not* selected, the Cut command will operate on the current line. This behavior can be controlled on the Configure | Preferences | Editing 1 options page. The option is titled *Cut/Copy/Append commands use current line when no text is selected*.

When a columnar selection (Block | Select Columnar) is placed on the clipboard, any under-length lines within the selected range will be extended with Spaces to match the width of the rectangular text block. This ensures that the block will behave as expected if the Paste command is later used to insert the text at a new location. Likewise, any Tab characters within the selected region will be converted to Spaces before being placed on the clipboard.

When operating in Typeover mode on a columnar selection, the Cut command will fill the selected area with Spaces without closing up the rectangle occupied by the text.

When placing columnar text onto a clipboard, Boxer must take care so that subsequent Paste operations of that text will be performed properly. Columnar clipboard text must be pasted differently than stream text, since all lines must move rightward by the width of the text block. Notwithstanding this fact, columnar clipboard text placed onto the Windows clipboard by Boxer can still be pasted into other Windows applications. Boxer does not use a *private clipboard format* for this purpose.

Boxer's clipboard commands will sense the type of text data being placed on the Windows clipboard and, when appropriate, use a more descriptive clipboard format to tag that data. For example, when Boxer senses that HTML code is being copied to the clipboard, the text will also be placed in the HTML clipboard format. This enables a conforming program to paste the data more intelligently. Boxer will also tag Rich Text data (RTF) and Comma-Separated Value (CSV).

5.65 Cut Append

Menu: Edit > Cut Append

Default Shortcut Key: Shift+Ctrl+X

Macro function: CutAppend()

The Cut Append command removes the selected text from the current file and adds it to the current clipboard. The current clipboard might be the Windows clipboard or one of Boxer's eight internal clipboards. See the Edit | Set Clipboard command for details on changing the current clipboard.

If text is *not* selected, the Cut Append command will operate on the current line. This behavior can be controlled on the Configure | Preferences | Editing 1 options page. The option is titled *Cut/Copy/Append commands use current line when no text is selected*.

When a columnar selection (Block | Select Columnar) is placed on the clipboard, any under-length lines within the selected range will be extended with Spaces to match the width of the rectangular text block. This ensures that the block will behave as expected if the Paste command is later used to insert the text at a new location. Likewise, any Tab characters within the selected region will be converted to Spaces before being placed on the clipboard.

When operating in Typeover mode on a columnar selection, the Cut Append command will fill the selected area with Spaces without closing up the rectangle occupied by the text.

Text cannot be appended to a clipboard if the selection type (stream or columnar) differs from the type of the text which is already present on the clipboard.


When placing columnar text onto a clipboard, Boxer must take care so that subsequent Paste operations of that text will be performed properly. Columnar clipboard text must be pasted differently than stream text, since all lines must move rightward by the width of the text block. Notwithstanding this fact, columnar clipboard text placed onto the Windows clipboard by Boxer can still be pasted into other Windows applications. Boxer does not use a *private clipboard format* for this purpose.


Boxer's clipboard commands will sense the type of text data being placed on the Windows clipboard and, when appropriate, use a more descriptive clipboard format to tag that data. For example, when Boxer senses that HTML code is being copied to the clipboard, the text will also be placed in the HTML clipboard format. This enables a conforming program to paste the data more intelligently. Boxer will also tag Rich Text data (RTF) and Comma-Separated Value (CSV).

5.66 Declaration

Menu: Jump > Declaration

Default Shortcut Key: none

Macro function: Declaration()

When editing within a supported source code file, the Declaration command provides a means to jump from an identifier reference to the point at which the identifier was declared. For example, when sitting at a function/procedure call, issuing the Declaration command will cause the cursor to jump to the declaration of the function/procedure being referenced. The Declaration command can also be used to jump to the declaration point of defined constants and global variables, so long as these entities are indexed by Ctags.

If the declaration resides in another file, that file will be opened and/or made current before moving the cursor to the relevant declaration line. If more than one declaration exists for the identifier at the cursor, the Ctags Function Index dialog will be displayed so that the proper instance can be selected.


The Declaration command relies upon the Ctags Function Index feature to perform its service. If the Ctags feature has been disabled, or if the file being edited is not supported by Ctags, the Declaration command will be unavailable.


After moving to a declaration, use the Reference command to return to the identifier reference from which the Declaration command was made.

5.67 Decrement

Menu: Edit > Math > Decrement

Default Shortcut Key: none

Macro function: Decrement()

The Decrement command can be used to decrement an integer value (ie, a whole number, not a floating point value) at the text cursor by another integer value. A dialog box will appear to retrieve the value to be subtracted. After clicking 'OK' the arithmetic is performed, and the old value is replaced by the result.

If the text cursor is situated on a character rather than a numeric value, the supplied value will be subtracted from the character at the cursor and the new character will be displayed. If the resultant character value is out of range, an error message will be given.

5.68 Delete (Text)

Menu: Edit > Delete

Default Shortcut Key: Del

Macro function: Delete()

The Delete command deletes the selected text from the current file. If text is not selected, the character at the text cursor is deleted.

When operating in Typeover mode on a columnar selection, the Delete command will fill the selected area with Spaces without closing up the rectangle occupied by the text.

5.69 Delete (Project)

Menu: Project > Delete

Default Shortcut Key: none

Macro function: ProjectDelete()

Use the Project | Delete command to delete a selected project file. This command deletes a project file. It does not delete the files named within that file.

☞ See the Project | New command for full details about Boxer's project file feature.

5.70 Delete Blank Lines

Menu: Edit > Delete > Blank Lines

Default Shortcut Key: none

Macro function: DeleteBlankLines()

The Delete Blank Lines command can be used to delete blank lines within the current file. If a range of lines is selected, the operation will be restricted to the selected lines. Due to the destructive nature of this command, a confirmation is required before the operation begins.

☞ A line is considered blank if it has no text, or if its text consists only of *whitespace*.

5.71 Delete Bookmarked Lines

Menu: Edit > Delete > Bookmarked Lines

Default Shortcut Key: none

Macro function: DeleteBookmarkedLines()

The Delete Bookmarked Lines command can be used to delete all bookmarked lines from the current file. For example, the Toggle Bookmark command can be used to 'flag' several lines for deletion, and then the Delete Bookmarked Lines command can be used to delete the lines.

This command deletes lines, not simply bookmarks. To remove the bookmarks from bookmarked lines, use either the Toggle Bookmark or the Bookmark Manager command. If you have accidentally deleted bookmarked lines when you meant only to clear their bookmarks, use the Undo command to recover these lines.

5.72 Delete Current Line

Menu: Edit > Delete > Current Line

Default Shortcut Key: Alt+D

Macro function: DeleteLine()

The Delete Current Line command deletes all of the text and the newline character on the current line. Whenever possible, the column of the text cursor will be preserved when moving to the next line.

5.73 Delete Duplicate Lines

Menu: Edit > Delete > Duplicate Lines

Default Shortcut Key: none

Macro function: DeleteDuplicateLines()

The Delete Duplicate Lines command can be used to delete duplicate lines within the current file. If a range of lines is selected, the operation will be restricted to the selected lines. Due to the destructive nature of this command, a confirmation is required before the operation begins.

Delete Duplicate Lines will not delete the first instance of a duplicated line. In other words, given a file that contained five lines with the text '`sample`', four of these lines would be deleted.


This command is similar in effect to the Find Unique Lines and Find Distinct Lines commands. For certain tasks, one of these commands might be more suitable.


To delete blank lines, use the Delete Blank Lines command.

5.74 Delete Lines that Begin with

Menu: Edit > Delete > Lines That Begin With

Default Shortcut Key: none

Macro function: DeleteLinesThatBeginWith()

This command can be used to delete all lines that begin with a user-specified text string. If a range of lines is selected, the operation will be restricted to the selected range.

The text string is supplied in a dialog box that appears after the command is issued. The "Match case" checkbox on that dialog can be used to control how the search is performed: with or without case sensitivity. Before the operation is performed, a second dialog box appears which tells the number of lines that will be deleted, and provides an opportunity to cancel the operation.

5.75 Delete Lines that Contain

Menu: Edit > Delete > Lines That Contain

Default Shortcut Key: none

Macro function: DeleteLinesThatContain()

This command can be used to delete all lines that contain a user-specified text string. If a range of lines is selected, the operation will be restricted to the selected range.

The text string is supplied in a dialog box that appears after the command is issued. The "Match case" checkbox on that dialog can be used to control how the search is performed: with or without case sensitivity. Before the operation is performed, a second dialog box appears which tells the number of lines that will be deleted, and provides an opportunity to cancel the operation.

5.76 Delete Lines that do not Begin with

Menu: Edit > Delete > Lines That Do Not Begin With

Default Shortcut Key: none

Macro function: DeleteLinesThatDoNotBeginWith()

This command can be used to delete all lines that **do not** begin with a user-specified text string. If a range of lines is selected, the operation will be restricted to the selected range.

The text string is supplied in a dialog box that appears after the command is issued. The "Match case" checkbox on that dialog can be used to control how the search is performed: with or without case sensitivity. Before the operation is performed, a second dialog box appears which tells the number of lines that will be deleted, and provides an opportunity to cancel the operation.

5.77 Delete Lines that do not Contain

Menu: Edit > Delete > Lines That Do Not Contain

Default Shortcut Key: none

Macro function: DeleteLinesThatDoNotContain()

This command can be used to delete all lines that **do not** contain a user-specified text string. If a range of lines is selected, the operation will be restricted to the selected range.

The text string is supplied in a dialog box that appears after the command is issued. The "Match case" checkbox on that dialog can be used to control how the search is performed: with or without case sensitivity. Before the operation is performed, a second dialog box appears which tells the number of lines that will be deleted, and provides an opportunity to cancel the operation.

5.78 Delete Lines that do not End with

Menu: Edit > Delete > Lines That Do Not End With

Default Shortcut Key: none

Macro function: DeleteLinesThatDoNotEndWith()

This command can be used to delete all lines that **do not** end with a user-specified text string. If a range of lines is selected, the operation will be restricted to the selected range.

The text string is supplied in a dialog box that appears after the command is issued. The "Match case" checkbox on that dialog can be used to control how the search is performed: with or without case sensitivity. Before the operation is performed, a second dialog box appears which tells the number of lines that will be deleted, and provides an opportunity to cancel the operation.

5.79 Delete Lines that End with

Menu: Edit > Delete > Lines That End With

Default Shortcut Key: none

Macro function: DeleteLinesThatEndWith()

This command can be used to delete all lines that end with a user-specified text string. If a range of lines is selected, the operation will be restricted to the selected range.

The text string is supplied in a dialog box that appears after the command is issued. The "Match case" checkbox on that dialog can be used to control how the search is performed: with or without case sensitivity. Before the operation is performed, a second dialog box appears which tells the number of lines that will be deleted, and provides an opportunity to cancel the operation.

5.80 Delete Next Word

Menu: Edit > Delete > Next Word

Default Shortcut Key: Ctrl+Del

Macro function: DeleteNextWord()

The Delete Next Word command deletes from the text cursor to the beginning of the next word.

The characters which serve to delimit words can be set on the Configure | Preferences | Cursor options page. The option is titled *These characters will delimit words.*

5.81 Delete Previous Word

Menu: Edit > Delete > Previous Word

Default Shortcut Key: Ctrl+Backspace

Macro function: DeletePreviousWord()

The Delete Previous Word command deletes from the text cursor to the end of the previous word.

The characters which serve to delimit words can be set on the Configure | Preferences | Cursor options page. The option is titled *These characters will delimit words.*

5.82 Delete to End of Line

Menu: Edit > Delete > to End of Line

Default Shortcut Key: none

Macro function: DeleteToEndOfLine()

The Delete to End of Line command deletes from the text cursor to the end of line. The Newline character is not deleted. The position of the text cursor is unchanged.

If issued on an empty line with the text cursor in column 1, the Delete to End of Line command will delete the entire line.

5.83 Delete to Start of Line

Menu: Edit > Delete > to Start of Line

Default Shortcut Key: Ctrl+K

Macro function: DeleteToStartOfLine()

The Delete to Start of Line command deletes from the left of the text cursor to the start of line. The cursor is left in Column 1.

If issued on an empty line with the text cursor in column 1, the Delete to Start of Line command will delete the entire line.

5.84 Divide

Menu: Edit > Math > Divide

Default Shortcut Key: none

Macro function: Divide()

The Divide command can be used to divide an integer value (ie, a whole number, not a floating point value) at the text cursor by another integer value. The result will be displayed as an integer value. A dialog box will appear to retrieve the value to divide by. After clicking 'OK' the arithmetic is performed, and the old value is replaced by the result.

If the text cursor is situated on a character rather than a numeric value, the character value of the character at the cursor will be divided by the supplied value and the resultant character will be displayed. If the resultant character value is out of range, an error message will be given.

5.85 Duplicate and Increment

Menu: Edit > Line > Duplicate and Increment

Default Shortcut Key: Shift+F2

Macro function: DuplicateAndIncrement()

The Duplicate and Increment command is similar to the Duplicate Line command, with one important difference: as it copies the current line to a new line below, it increments any values it finds within the line. A few examples will help illustrate its utility:

When the cursor is placed on a line with the following text:

```
width1 = MainForm->WidthArray[1];
```

and the Duplicate and Increment command is issued three times, the following text will result:

```
width1 = MainForm->WidthArray[1];
```

```
width2 = MainForm->WidthArray[2];
width3 = MainForm->WidthArray[3];
width4 = MainForm->WidthArray[4];
```

Duplicate and increment also recognizes character constants...

```
char01 = 'A';
```

would become:

```
char01 = 'A';
char02 = 'B';
char03 = 'C';
char04 = 'D';
```

... and on hexadecimal values:

```
pos[15] := $DF;
```

becomes:

```
pos[15] := $DF;
pos[16] := $E0;
pos[17] := $E1;
pos[18] := $E2;
```

Hexadecimal values are recognized in three forms: 0xFF, FFh and $FF.

The examples above relate to programming, but Duplicate and Increment can also be useful in non-technical situations. If you needed to start a numbered list of items, you could create the first line:

```
Part No. 3141001
```

and then use Duplicate and Increment to make as many copies as needed:

```
Part No. 3141001
Part No. 3141002
Part No. 3141003
Part No. 3141004
Part No. 3141005
```

When the Duplicate and Increment command is issued repeatedly to duplicate a line, the status line will report a count of the number of times the command has issued.

5.86 Duplicate Line

Menu: Edit > Line > Duplicate Line

Default Shortcut Key: F2

Macro function: DuplicateLine()

The Duplicate Line command can be used to make a copy of the current line. The new line will be created below the current line, and the text cursor will be moved onto the new line at the current cursor column.

The Duplicate Line command is a quick alternative to using the Copy and Paste commands to perform the same task, and does not affect the content of the current clipboard.

When the Duplicate Line command is issued repeatedly to duplicate a line, the status line will report a count of the number of times the command has issued.

5.87 EBCDIC to ASCII

Menu: Block > Convert Other > EBCDIC to ASCII

Default Shortcut Key: none

Macro function: EBCDICtoASCII()

This command will convert text encoded in the EBCDIC character set to the ASCII character set. The text to be converted must first be selected. If an entire file is to be converted, use the Select All Text command to select the whole file.

EBCDIC is a character encoding system used primarily on mainframe computers. The ASCII character encoding system is used widely on personal computers. At times, a file that uses EBCDIC encoding may need to be converted for use on a computer that uses the ASCII character encoding system. This command can be used for that purpose.

EBCDIC is an acronym for Extended Binary Coded Decimal Interchange Code.

ASCII is an acronym for American Standard Code for Information Interchange.

5.88 Edit Active

Menu: Project > Edit Active

Default Shortcut Key: none

Macro function: ProjectEditActive()

Use the Edit Active command to open the active project file for editing. This command opens a project file as a text file; it does not open the files named within that project file. To open the files within a project file, use the Project | Open command.

See the Project | New command for full details about Boxer's project file feature.

5.89 Edit Clipboard

Menu: Edit > Edit Clipboard

Default Shortcut Key: none

Macro function: EditClipboard()

The Edit Clipboard command can be used to edit the content of the clipboard selected. The content of the clipboard is placed into an editing window and can be edited in all the same ways a file may be edited. When the text in the window is saved, it is written back to the clipboard. The Windows clipboard and any of Boxer's internal clipboards can be edited. The Windows clipboard is not eligible for editing if it contains non-text data.

The content of each clipboard is displayed in a popup window as the menu cursor is moved across the clipboard's menu entry. This makes it easy to check what's on a clipboard without pasting the content into a file or opening it for editing.

The content of Boxer's internal clipboards will be saved at the end of an edit session, as long as the length of the text on the clipboard is 2,048 characters or less. Because the content of the internal clipboards persists from session to session, and cannot be changed by other applications, these clipboards can be useful for storing frequently used text blocks for insertion into your files. The Edit Clipboard command might be used to create these text blocks and maintain them.

The content of a clipboard can be cleared with the Clear Clipboard command. The content of *all* clipboards can be cleared with the Clear All Clipboards command.


When the content of a clipboard is displayed in a popup window, the text is displayed with an 8 point, fixed width, *Courier New* font. This font utilizes the ANSI character set mapping. If the current screen font uses an OEM character set mapping, and if characters outside the normal alphanumeric range reside on the clipboard, then the content of the clipboard may appear different in the popup window than it would in the underlying file. This difference is simply the result of a difference in character sets, and does not mean that the data on the clipboard has been adjusted or corrupted.

5.90 Edit Other

Menu: Project > Edit Other

Default Shortcut Key: none

Macro function: ProjectEditOther()

Use the Edit Other command to open a selected project file for editing. This command opens a project file as a text file; it does not open the files named within that project file. To open the files within a project file, use the Project | Open command.

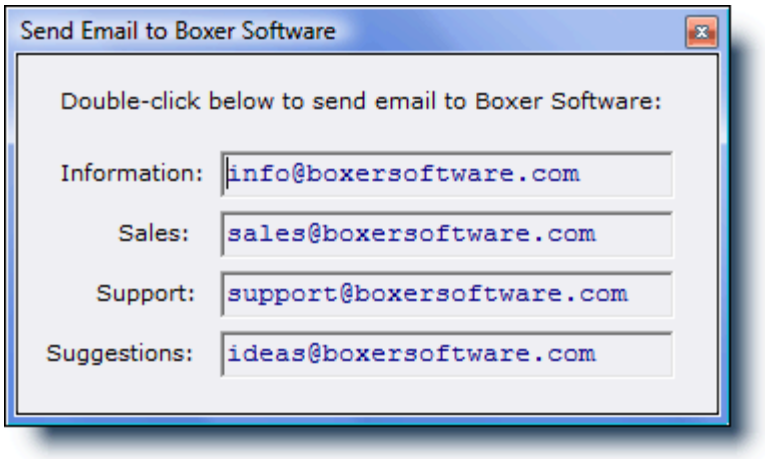
☞ See the Project | New command for full details about Boxer's project file feature.

5.91 Email Boxer Software

Menu: Help > Email Boxer Software

Default Shortcut Key: none

This command displays a dialog box with email addresses which can be used to contact us at Boxer Software. Just click on the email address that fits your need, and your email client will be run so that a message can be sent.



☞ In order to launch your email program, Boxer relies upon the operating system shell's ability to process a 'mailto' directive. When an email client program is installed, it typically establishes itself as the program which is called by the shell to process the 'mailto' directive. If you find that your active email program is not launched by Boxer, or if some other inactive email program is launched instead, it's probably because your active email program did not establish itself to be the program that processes mailto commands. This situation cannot be remedied by Boxer, and is not due to any shortcomings in Boxer. You might consult the documentation of your email program or contact its vendor.

5.92 Error Chart

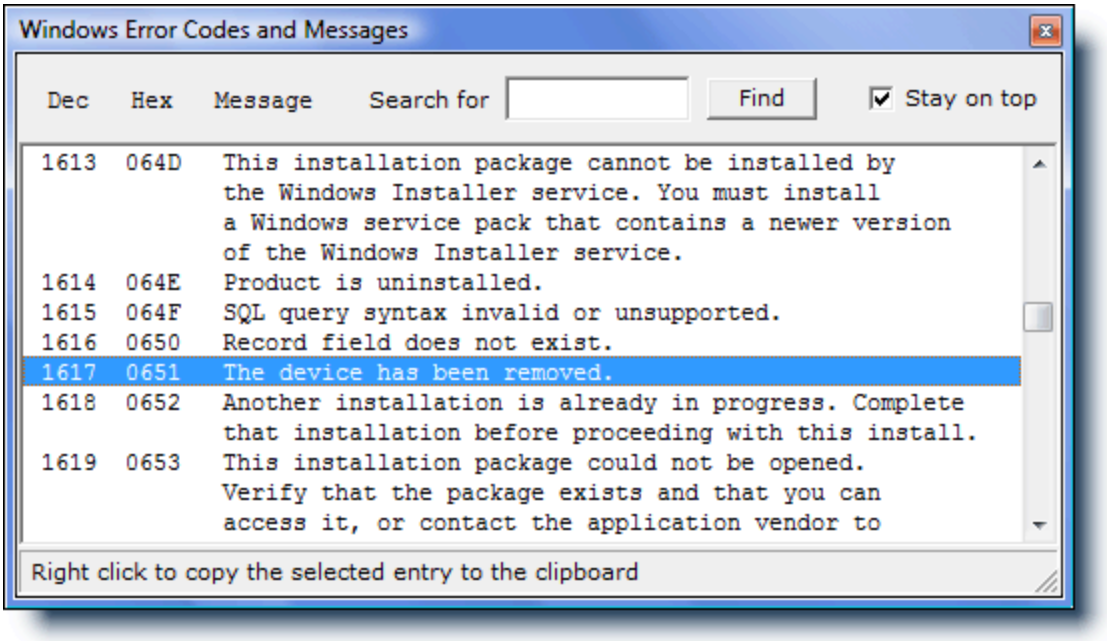
Menu: Tools > Error Chart

Default Shortcut Key: none

Macro function: ErrorChart()

The Error Chart command displays a popup list of Windows error codes and their associated messages. When errors are reported by the operating system--or by an

application program--they will often reference a numeric error code. These reports frequently have insufficient information about the error which occurred. Boxer's Error Chart can be used by programmers--or by any users--to decode the meaning of Windows error codes.



The Error Chart can be searched by value or by any text which appears within the listing. Type the search string into the edit box provided and click *Find*. The *Find* button is also used to find the next occurrence of a string which has just been found.

Right-clicking on a selected item summons the Error Chart context menu. The context menu provides an option to copy the selected message to the current clipboard.

If you prefer that the Error Chart remain atop other windows, select the *Stay on top* option. The Error Chart is a *non-modal* window, which allows it to remain on-screen after *focus* has been returned to another editing window.

If the Error Chart is left on-screen when Boxer is closed, it will be automatically reopened if the edit session is later restored.

5.93 Exit

Menu: File > Exit

Default Shortcut Key: Alt+F4

Macro function: Exit()

The Exit command is used to close Boxer and end your editing session. If unsaved changes have been made to any file a dialog box will appear for each such file to alert you to this condition. You will then be able to choose whether to save the changes

before exiting, exit without saving, or cancel the Exit request altogether.

You can quickly tell whether a file has unsaved changes by looking for an asterisk (*) to the left of its name in the title bar, or on its File Tab.

Boxer has an option to warn on exit if the size of the text on the clipboard exceeds a user-defined threshold. This option can be found on the Configure | Preferences | Messages dialog.

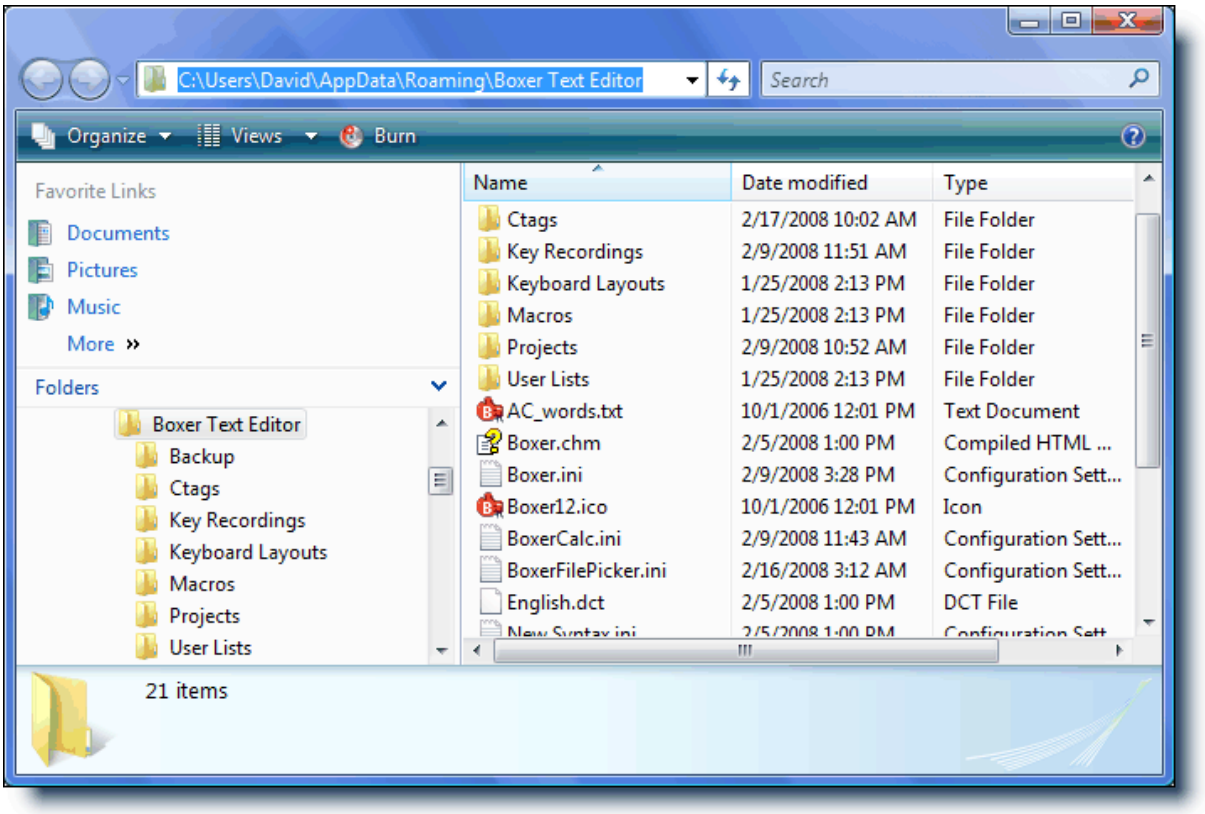
5.94 Explore Data Folder

Menu: Configure > Explore Data Folder

Default Shortcut Key: none

Macro function: ExploreDataFolder()

This command provides a convenient method of opening an Explorer window that points to the folder that holds Boxer's application data files. Among these are the files that contain syntax highlighting information, templates, user lists, projects, macros, as well as others.



See also the Explore Program Folder command.

☞ When installed on an operating system prior to Windows Vista, Boxer's

home/installation folder serves double duty as both its program folder and its data folder. Beginning with Windows Vista, and also under Windows 7, Boxer will maintain separate folders for its program and data files.

This command can be used to quickly locate the folder that contains the backup files Boxer creates. This folder should be emptied periodically to conserve disk space.

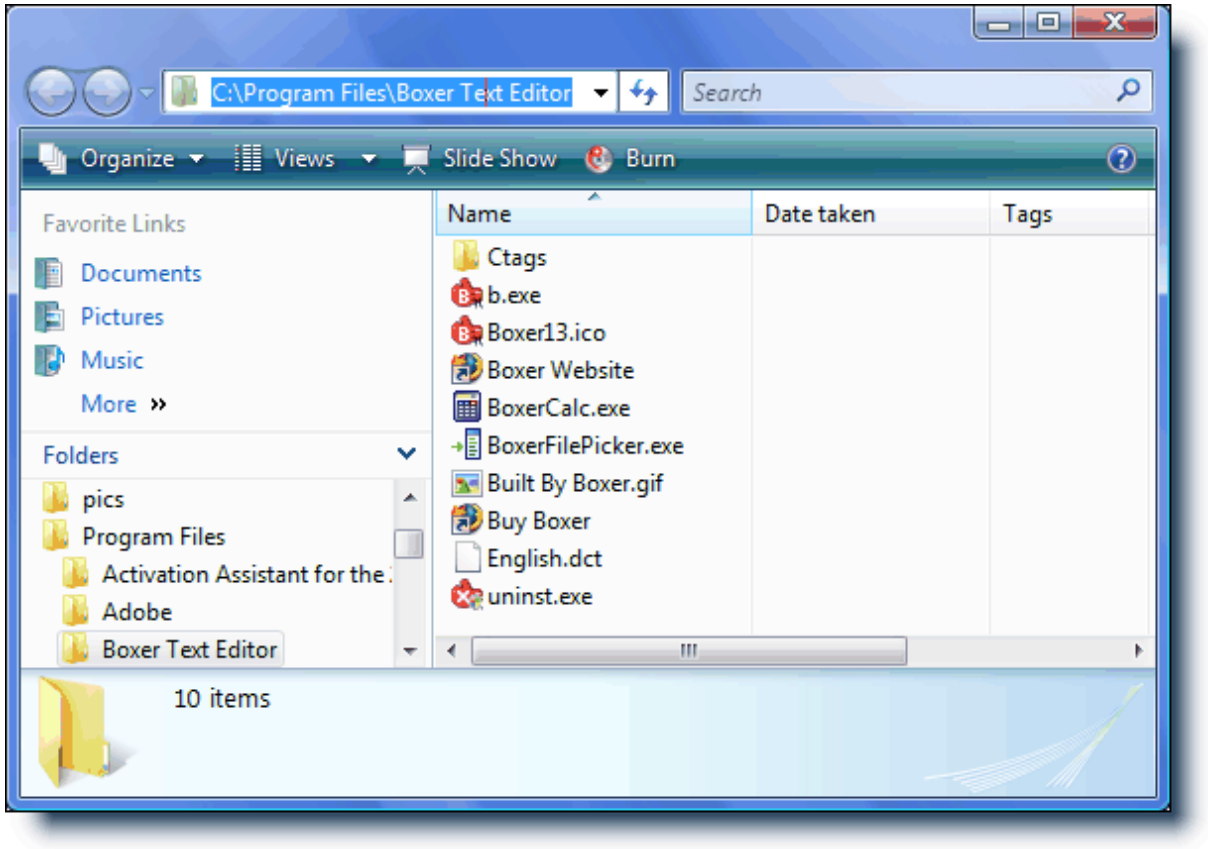
5.95 Explore Program Folder

Menu: Configure > Explore Program Folder

Default Shortcut Key: none

Macro function: ExploreProgramFolder()

This command provides a convenient method of opening an Explorer window that points to the folder that holds Boxer's program files. Among these are the Boxer program itself, calculator, uninstaller, icons and other supporting files.



See also the Explore Data Folder command.

When installed on an operating system prior to Windows Vista, Boxer's home/installation folder serves double duty as both its program folder and its data folder. Beginning with Windows Vista, and also under Windows 7, Boxer will maintain separate folders for its program and data files.

5.96 FAQs

Menu: Help > FAQs

Default Shortcut Key: none

Frequently Asked Questions

After I order, will I get a password or key to convert the evaluation version of Boxer into a licensed version?
No. New software will be sent which is easily installed atop the evaluation version. All of your settings will be maintained. Software keys are frequently distributed on 'pirate' Internet sites, thus reducing sales and driving up the cost of software for paying customers.

Why can't other programs see the text I copied to the clipboard in Boxer?
You are almost certainly using an internal clipboard, rather than the Windows clipboard. Other programs can't see text that is placed on Boxer's internal clipboards. See the Set Clipboard command for details.

Why am I having trouble opening filenames from Explorer when they contain embedded spaces?
This is due to a bug in Explorer. It doesn't enclose a filename in double quotes before sending it off to the associated application. In the file associations set up by Boxer's installer, double quotes have been added around "%1", so you'll find these associations (.TXT, .BAT, etc) work fine. But for any file associations you create yourself, or if you elected not to allow Boxer's installer to create the associations, you'll need to manually edit the association to have double quotes around "%1". You'll find that Boxer's help topic entitled 'File Associations' has additional useful information about this subject.

Why can't I see all my Windows fonts in the Screen Font dialog?
Boxer requires that *fixed width* fonts (monospace fonts) be used, so the Screen Font dialog box does not display *proportionally spaced* fonts. This is required, in part, to ensure that columnar selections can be highlighted neatly in rectangular blocks, and so that the Column Ruler can be used. These features would not be possible if the use of proportionally spaced fonts was permitted.

Will there be a German version of Boxer for Windows, as there was for earlier Boxer products?
It appears unlikely. We learned from our earlier products that the effort to release a program in a new language is quite substantial. It appears that our resources can be better spent enhancing our current products, or developing new ones.

Will there ever be a Linux version of Boxer?

That's uncertain at this time. We're keeping an eye on the Linux market, and will continue to do so.

How long did it take to develop Boxer for Windows?
The initial development took almost two years. Boxer for Windows was a ground-up effort, with almost none of the code from our earlier products being used in its development.

What language was Boxer written in? How many lines of code? What development tool was used?
Boxer currently consists of over 110,000 lines of C++ code. Borland's C++ Builder was used for development.

Where did the name 'Boxer' come from?

In the mid 1980's, one of the most popular editors for the PC was a product called BRIEF, which was then marketed by a company called UnderWare. In fact, the very first lines of Boxer/DOS were written using Brief, until Boxer was able to edit its own code. The name Boxer was simply a play on words: another style of men's underwear!

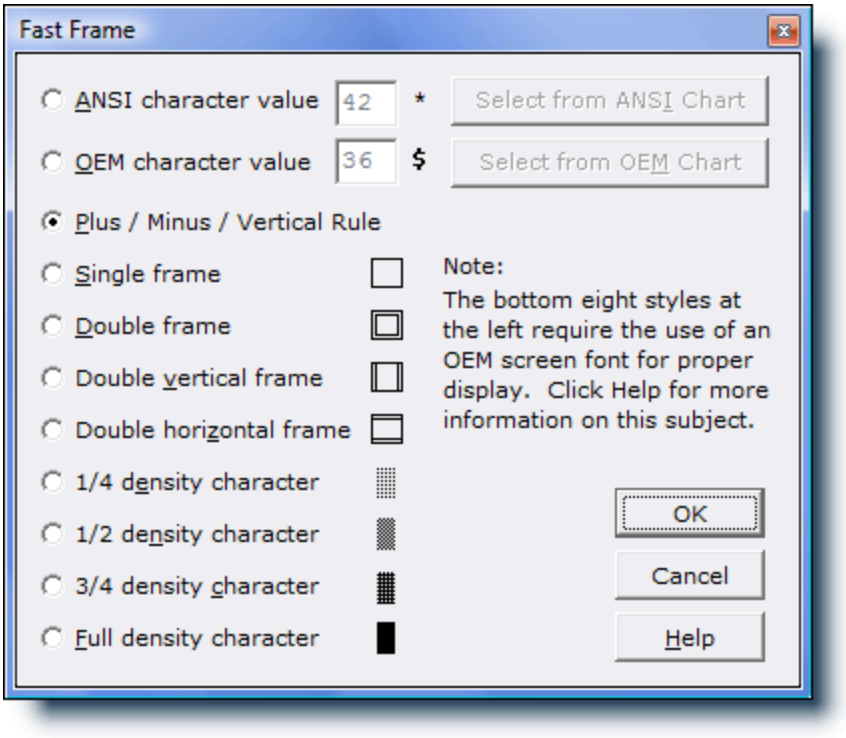
5.97 Fast Frame

Menu: Tools > Fast Frame

Default Shortcut Key: Alt+F12

Macro function: FastFrame()

The Fast Frame command can be used to frame a columnar-selected rectangle with a chosen frame style, or with a specified character. A dialog box appears from which the frame style is selected:



A frame style is selected by checking the radio button which corresponds to the desired style. Options are also provided to frame the selected area with some other character, and either the ANSI Chart or OEM Chart can be summoned to assist in character selection.

When using an OEM Screen Font, several additional frame styles may be used. The bottom eight styles offered in the dialog box require the use of an OEM screen font for proper display. The ANSI character set does not contain these characters, so using these styles with an ANSI Screen Font will produce undesirable results.

Once OK is clicked, the selected area is automatically framed with the chosen frame style. The frame is applied to the outside of the selected area. If the left edge of the selection lies in column one, the lines in the selected range will be pushed right by one column to make room for the frame. If Tab characters appear within the selection, they will be automatically converted to Spaces to ensure proper display after framing.

When printing files which contain drawing characters from the OEM character set, be sure to use a Printer Font which also uses the OEM character set.

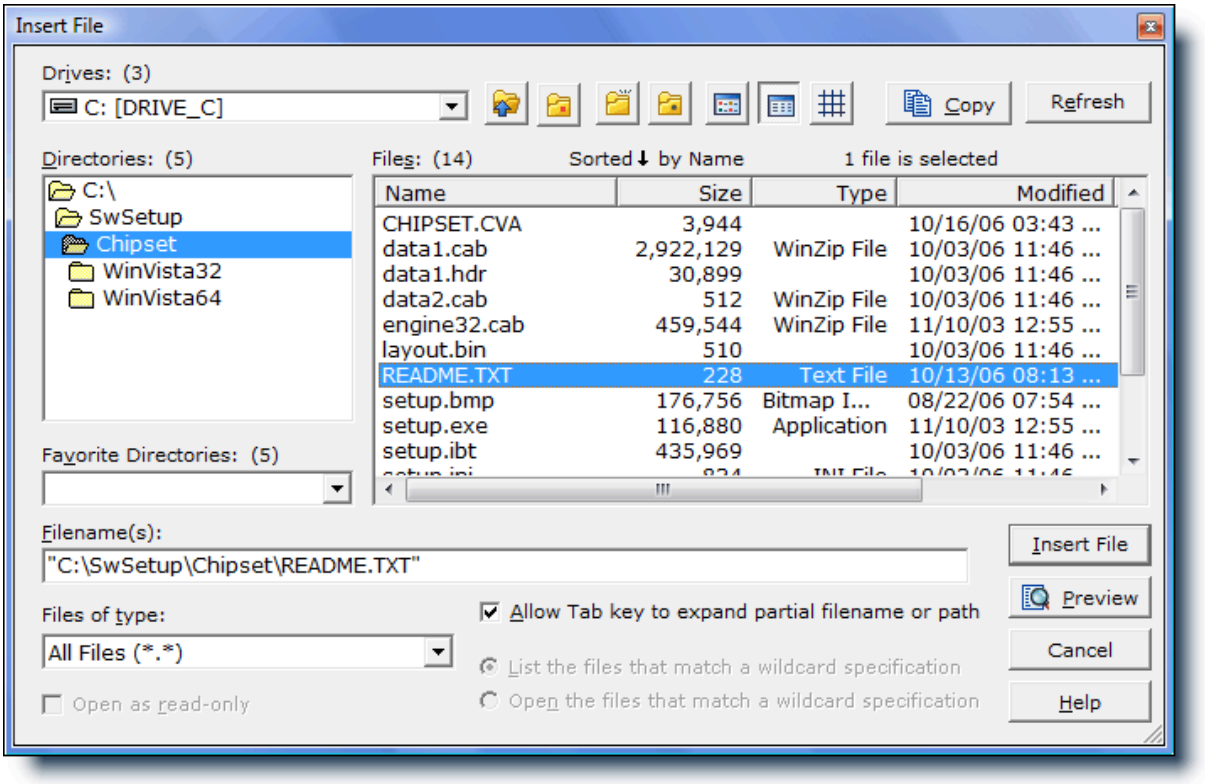
5.98 File Insert

Menu: File > Insert

Default Shortcut Key: Ctrl+I

Macro function: InsertFile()

The File Insert command is used to insert (some may say 'import') the content of another file at the current location of the text cursor. The file open dialog is presented for selecting a file and, after its selection, the content of the file will be placed at the cursor.



Any text file can be selected for insertion (subject to Sizes and Limits), even one which is being edited in another editor window.

See the Open command for full details on using both the custom and standard Windows File Open dialogs.

5.99 File Picker

Menu: File > Picker

Default Shortcut Key: Alt+K

Macro function: FilePicker()

The File Picker command opens a dockable tool window alongside Boxer that can be used to open files for editing. The File Picker can remain open while Boxer is in use, making it easy to open new files for editing whenever the need arises. The treeview interface can display files from either the local PC or from PCs on attached networks. The File Picker also contains logical entries for "Desktop," "My Documents," and other conceptual locations on the local PC.

To open a file, double click on its name, or select it and press *Enter*. To open multiple files, select the files of interest and press *Enter*.

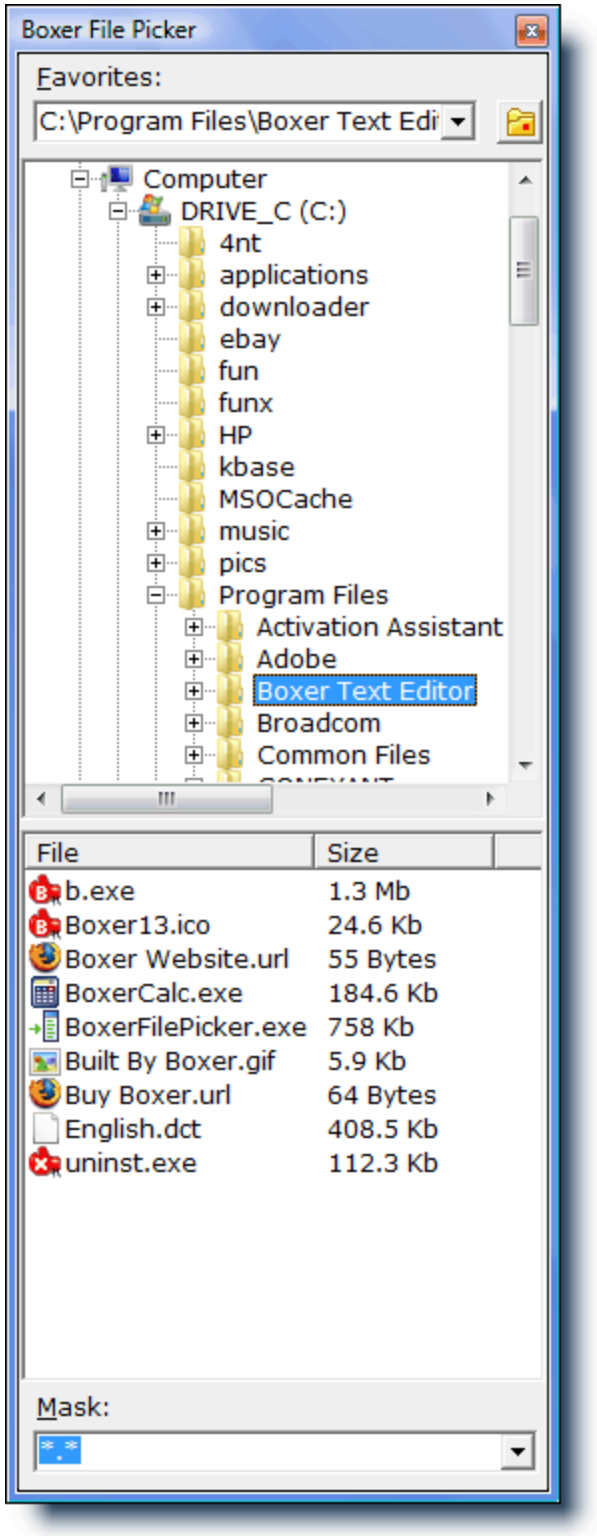
The right-click context menu contains options to open the selected file in its default application, or with an application of your choice. Commands to Cut, Copy, Rename and Delete files are also provided. Options are also provided to close the File Picker automatically when a file is opened, and/or when Boxer itself is closed.

A list of recent directories from which files have been opened is maintained in the *Favorites* list at the top of the File Picker window.

The *Mask* control permits the display of files to be filtered to show only a particular class of files.

The column headers in the file display area permit the file listing to be sorted by filename or file size.

By dragging the horizontal divider bar between the directory and file panes, you can control how much space is allocated to each pane.



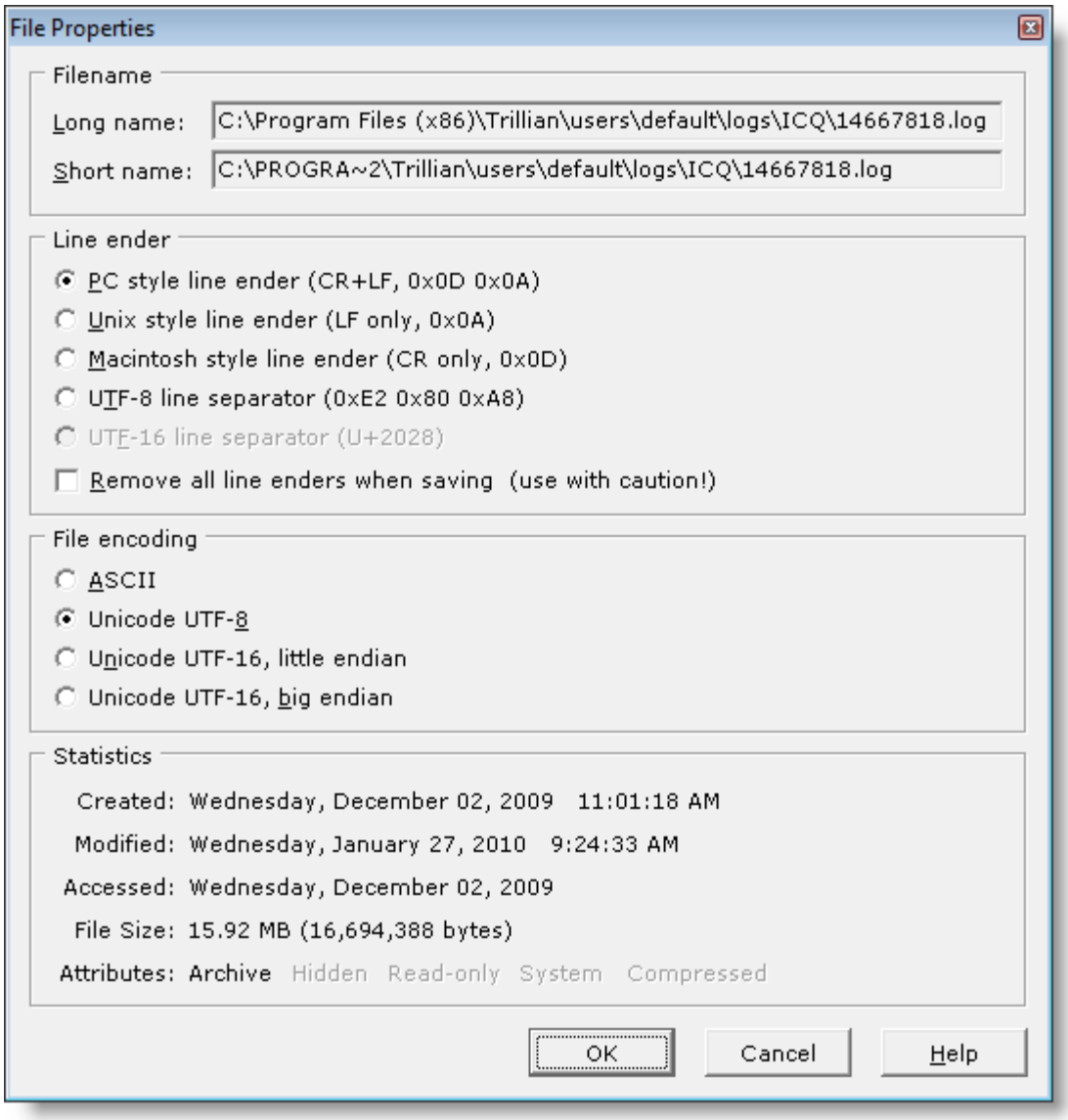
5.100 File Properties

Menu: File > Properties

Default Shortcut Key: none

Macro function: FileProperties()

The File Properties command displays a dialog box containing information about the file on disk which is associated with the editor's current file. The file's long and short names, create/modify/access times, file size and file attributes are all displayed. Because this command reports information about the current file's disk file, it will be disabled when editing a file which has yet to be written to disk.



Line ender

PC
Use this option to save files with a PC style line ender (CR+LF).

Unix
Use this option to save files with a Unix style line ender (LF only).

Macintosh
Use this option to save files with a Macintosh style line ender (CR only).

UTF-8 line separator
Use this option to save files with a UTF-8 line separator. This option is only valid when file encoding is set to Unicode UTF-8. Note that this line ender sequence uses three bytes: hex 0xE2, 0x80 and 0xA8.

UTF-16 line separator
Use this option to save files with a UTF-16 line separator (U+2028). This option is only valid when file encoding is set to one of the Unicode UTF-16 encoding options.


Boxer can read files with any type of line ender, and it will make note of the line ender as the file is read. The above options can be used to force a file to be written with a specific line ender.


An option to set the line ender styles that is used for newly created files appears on the Configure | Preferences | Editing 1 dialog page.

Remove all line enders when saving
This option can be used to ensure that no line enders are written to the output file when it is saved. This option should **not** be used for conventional text files, as all line-related formatting will be lost. Rather, this option can be useful when saving a file that contains fixed length records, if its file format requires that line enders not appear within the data stream. To load such a file for viewing or editing, Boxer's File Open dialog contains an option to impose a fixed length record format onto the file being opened. The -F command line option is also available for this purpose.

File encoding

ASCII
Use this option to save the current file with ASCII encoding. This is the standard file format for most text files. No header characters or null characters will appear in the output file.

UTF-8
Use this option to save the current file with UTF-8 encoding. This is a Unicode format that uses between one and four bytes to encode each character unambiguously. Null characters do not appear in UTF-8 encoded files.

Though it is not displayed on-screen in the editor, the three-byte UTF-8 Byte Order Mark (0xEF 0xBB 0xBF) will be applied when a UTF-8 encoded file is saved to disk. These bytes will appear as the first three bytes in the file.

UTF-16, little endian
Use this option to save the current file with UTF-16 little endian encoding. This is a

Unicode format that uses two bytes to encode each character that appears in the file. The 'little endian' designator refers to the byte order in the output file. The little endian version of UTF-16 is popular on Windows-based PCs. Null characters will almost certainly appear in the output file.

Though it is not displayed on-screen in the editor, the UTF-16 Byte Order Mark (U+FEFF) will be applied when a UTF-16 encoded file is saved to disk. In a UTF-16 little endian file, the first two bytes will be 0xFF 0xFE.

UTF-16, big endian
Use this option to save the current file with UTF-16 big endian encoding. This is a Unicode format that uses two characters to encode each character that appears in the file. The 'big endian' designator refers to the byte order in the output file. Null characters will almost certainly appear in the output file.

Though it is not displayed on-screen in the editor, the UTF-16 Byte Order Mark (U+FEFF) will be applied when a UTF-16 encoded file is saved to disk. In a UTF-16 big endian file, the first two bytes will be 0xFE 0xFF.


See the help topic Unicode Files for full information about Boxer's handling of Unicode files.


The active *code page* can be viewed using the System Info option on Boxer's About dialog.


An option to set the default line ender for newly created files appears on the Configure | Preferences | Editing 1 dialog page


An option to set the file encoding format for newly created files appears on the Configure | Preferences | Editing 1 dialog page

Statistics

The statistics section displays the date and time when the current file was created/modified/accessed, as well the file size and file attributes. When a file attribute is set, it is displayed in normal density; when not set, it is displayed as grayed or disabled.


The Long Name and Short Name properties are displayed in read-only edit boxes to permit the strings to be copied to the Windows clipboard, if desired.

5.101 File Tabs

Menu: View > File Tabs > View File Tabs

Default Shortcut Key: none

Macro function: ViewFileTabs()

The View File Tabs command is used to toggle on and off the display of the *File Tabs* which can appear at the bottom or top of Boxer's window. File Tabs provide a convenient method of switching among the currently open windows.

The tab for the current file is displayed as the uppermost tab, and its name will appear in **bold** text. For example, in the picture above, the current file is `REPORT.C`. Clicking on any other tab will bring that file to the foreground position.

Right clicking on an open area of the File Tab bar will provide access to its *context menu*, which allows the bar to be repositioned or turned off. The File Tab bar can also be repositioned by dragging it to a new location.

The context menu also contains an option to sort the File Tabs alphabetically, by filename. The order of the File Tabs controls the behavior of the Window Previous and Window Next commands.

An asterisk (*) is placed in front of the filename on the File Tab to indicate that the file has changes which have not yet been saved to disk.

- To reorder the file tabs, use the mouse to drag a file tab to a new location and drop it. When an edit session is resumed, the position of the file tabs will be maintained. The position of file tabs is also maintained within a project file. Note: repositioning file tabs by drag-and-drop necessitates that any file tab sorting mode (name, extension or use) which may be in force be abandoned. Otherwise, when a new file is opened and the file tabs are resorted, the drag-and-drop ordering would be lost.

- The filename displayed on the File Tab can be shortened to a user-defined width. This option appears on the Configure | Preferences | Display options page.

- A file can be closed by clicking its File Tab with the middle mouse button.

5.102 File Tabs - Bottom

Menu: View > File Tabs > Bottom

Default Shortcut Key: none

Macro function: FileTabsBottom()

Use this command to cause the file tabs to be located at screen bottom.

- The File Tab context menu also includes options to place the file tabs at screen top or bottom.

5.103 File Tabs - Top

Menu: View > File Tabs > Top

Default Shortcut Key: none

Macro function: FileTabsTop()

Use this command to cause the file tabs to be located at the top of the screen.

The File Tab context menu also includes options to place the file tabs at screen top or bottom.

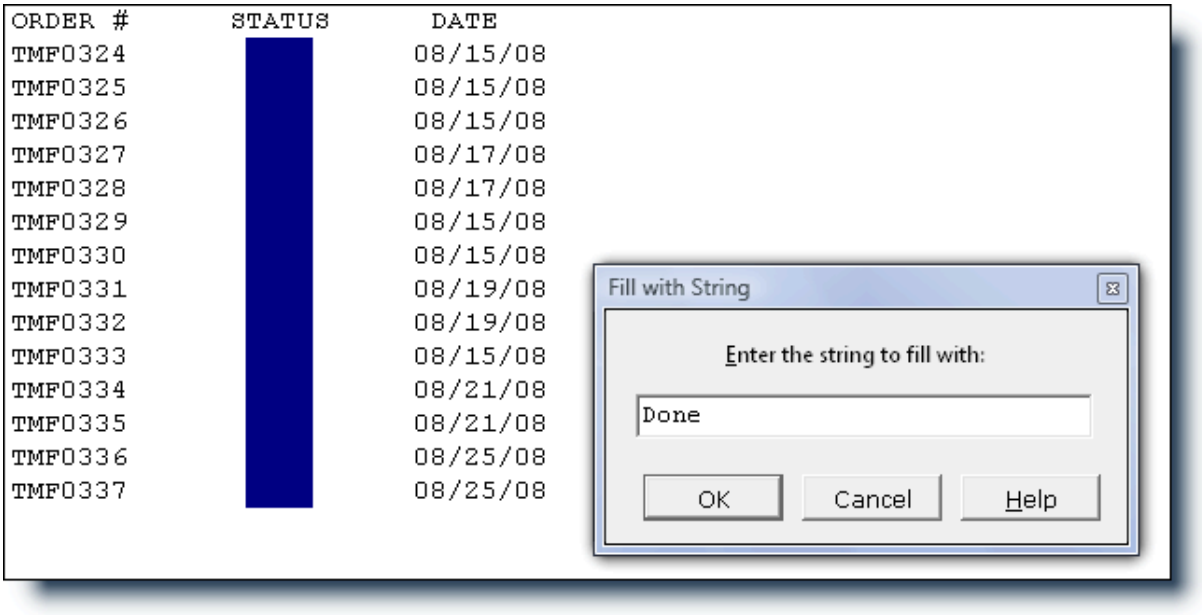
5.104 Fill with String

Menu: Block > Fill with String

Default Shortcut Key: none

Macro function: FillWithString()

The Fill with String command can be used to fill a selected area with a supplied text string. The selected area will be filled automatically with the string supplied. If the text supplied is less than the width of the selected text, the pattern will be repeated to fill the selected area.



To prevent unexpected results, the lines affected are automatically de-tabbed prior to performing the Fill operation. If any lines in the selected range are too short, they will be extended so that a complete fill of the selected area is achieved.

Special characters can be entered into the Fill with String edit box using the technique described in the Help topic Inserting Special Characters.

5.105 Find

Menu: Search > Find

Default Shortcut Key: Ctrl+F

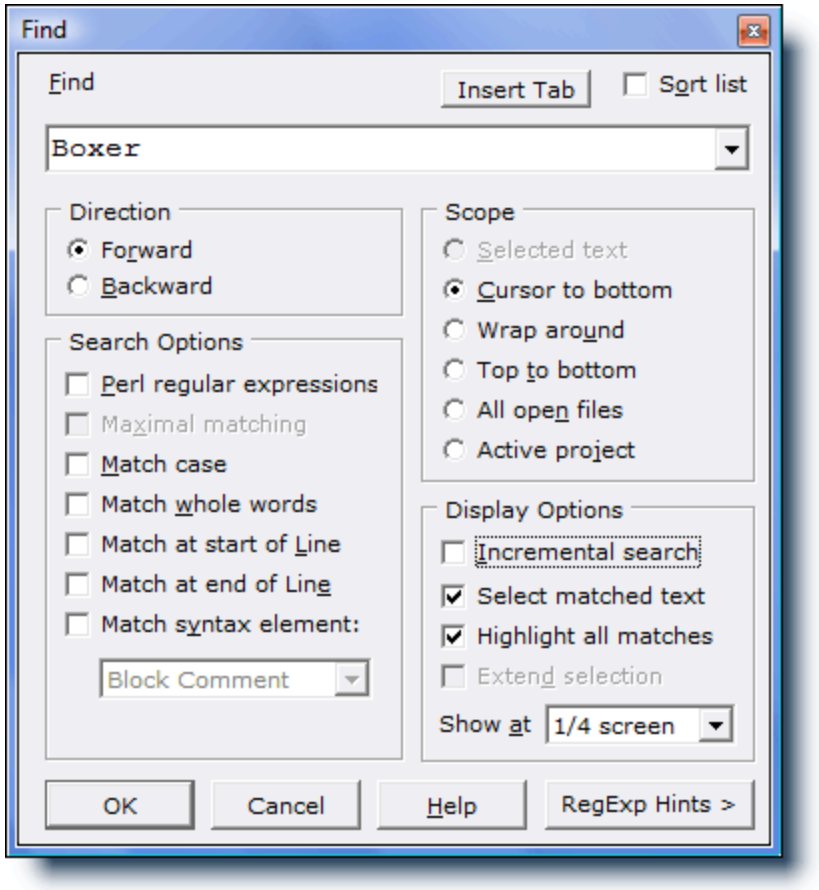
Macro function: Find()

The Find command is used to specify and initiate a search for a text string. Many different options are available to make searching more flexible and more powerful. Wildcard characters (also known as Regular Expressions) can also be used within the search string.

If the search string is found the text cursor will be moved to the matching string and the text will be selected, if the *Select matched text* option is active. The matched text can then be operated upon as can any other selected text.

If the search string is not found a dialog box will appear to report this fact. If you prefer that this report appear on the message line instead, use the option provided on the Configure | Preferences | Messages options page. The option is titled *Report failed searches in a popup message box*.

The controls and options in the Find dialog box are described below:



Find
This is the edit box where the search string is entered. When the Find command is issued, the word beneath the text cursor is placed into the *Find* edit box, in case that word--or a word which is nearly the same--is to be the search string. The Find Fast command can also be used to search for the word at the text cursor without raising the Find Text dialog. To recall a search string which was previously entered, use the drop-down list or press the up or down arrow keys to review the items in the history list. Regular Expressions may be used within the search string.

The *Delete* key can be used while the drop-down list is displayed to delete a selected entry from the history list.

Special characters can be entered into the *Find* edit box using the technique described in the Help topic Inserting Special Characters.

Insert Tab
Use this button to insert a tab character into the *Find* edit box.

Ordinarily, the *Tab* key is used to move from field to field within a dialog box. If you would prefer that the Tab key insert a tab character in this dialog box, and in other Find/Replace related dialog boxes, check the relevant box on the Configure | Preferences | Tabs dialog page.

☞ In most fonts, the tab character does not have a unique visual representation. It will often be depicted as an open square box (□), as will be other characters in the low-ASCII portion of the character set.

Sort list
If this box is checked the history list will be maintained in alphabetic order, rather than in the order the strings were entered.

☞ When switching to an alphabetically sorted list, the chronological ordering of the list will be lost, and cannot be restored by unchecking the checkbox.

☞ No attempt is made to associate the history list entries with the time that they were added to the list. If a sorted history list is used consistently, over time the list will come to hold an unrepresentative set of search phrases. In the extreme case, after many Find operations, a list could result that contained only phrases beginning with the letter 'A'. This occurs because entries at the bottom of the list will be removed after the maximum size of the list is reached.

Direction

Forward
This option causes the search to be performed downward, toward the end of file.

Backward
This option causes the search to be performed upward, toward the start of file.

Search Options

Perl regular expressions
If this box is checked, wildcard characters within the search string will be interpreted according to the Perl-Compatible Regular Expression (PCRE) convention. In part, this means that the asterisk (*) will cause a match of zero or more occurrences of the preceding character. The period (.) will match any single character. For more information, see Regular Expressions.

Maximal matching
When using pattern matching characters, there can sometimes be more than one text string that matches the search string. This option can be used to request that the longest possible matching string be returned.

Match case
This option can be used to force the search string to be matched exactly. When unchecked, a case insensitive search is performed.

Match whole words
This option can be used to restrict matches to those strings which appear as a whole word. The characters which serve to delimit words are user-configurable; see Configure | Preferences | Cursor.

☞ The Match Whole Words option is logically incompatible with the Incremental Search option, and will therefore be disabled when Incremental Search is active.

Match at start of Line
This option can be used to force the search string to be matched only when a matching string appears at the start of a line. This effect can also be achieved with a Regular Expression.

Match at end of Line
This option can be used to force the search string to be matched only when a matching string appears at the end of a line. This effect can also be achieved with a Regular Expression.

Match syntax element
This option can be used to force a matching string to belong to a specified syntax group. Example: you could type the search string 'while' and require that it be matched only when it appears as a Reserved Word. Instances that occur within program comments or quoted strings (or any other syntax) would not be matched. (This is a powerful capability with lots of potential, and one we've never seen in another editor.)

Scope

Selected Text
This option can be used to restrict the search to the extent of the selected text.

Cursor to bottom / Cursor to top
This option causes the search to be performed from the text cursor onward, according to the current direction. The search ends when either the top or bottom of the file is reached.

Wrap around
This option causes the search to be performed from the cursor onward, according to the current direction. When either the top or bottom of file is reached, the search wraps around, and continues to the original cursor position.


When Find Next or Find Previous are used in wrap around mode, a message appears on the status bar when the search has wrapped back to the location of the first match. An option on the Configure | Preferences | Messages dialog page can be used if you prefer that this event be reported in a pop-up message box instead.

Top to bottom / Bottom to top
This option causes the search to be performed from the top or bottom of file onward, according to the current direction.

All open files
This option causes the search to be performed across all open files.

Active project
Use this option to limit the scope of the Find operation to those files within the active project.

Display Options

Incremental search
This option causes the search process to begin as soon as a character is pressed, rather than waiting for OK to be pressed. When typing long search strings, you may find that the match is found before you're done, thereby saving typing. Just press *Enter* to dismiss the dialog and remain at the displayed match.

This option is disabled when Perl Regular Expressions are in use, since it is often the case that a regular expression cannot be properly evaluated until it has been completely typed.

Select matched text
This option causes the matched string to be selected so that it can be operated upon by any command that operates upon selected text. When this option is not used, the text cursor is simply placed at the start of the matching string.

This option is incompatible with the *Extend Selection* option, and will therefore be disabled when that checkbox is checked.

Highlight all matches
This option causes all instances of the matched string to be highlighted within the current file, and within other edited files. The highlighting will persist until a new Find operation is performed, or until the end of the editing session. Alternatively, highlighting can be disabled altogether using the View | Text Highlighting command (this also affects the general Text Highlighting feature). The foreground and background colors used to highlight matches can be set on the Configure | Colors dialog.

This option is not available for searches which use Regular Expressions.

To permanently configure one or more text strings for highlighting, use the Text Highlighting feature.

Extend Selection
This option can be used to extend an existing selection to the point of the matched string.

This option is logically incompatible with the *Select matched text* option, and will therefore be disabled when that checkbox is checked. This option will also be disabled when the *Scope* has been set to *Selected text*.

Show at
This option controls the screen position at which matched strings are displayed. When a new match is already on-screen, it will be shown in place, without redrawing the screen. If the screen must be redrawn to show a new match, then the matching line will be positioned at the designated screen location.

Notes

When the *Incremental search* and *Select matched text* options are both in use, the Find Next and Find Previous commands can be issued from the keyboard (*F3* and *Shift+F3*, respectively) in order to display additional matches to the partially typed

search string while the Find dialog is still open.

5.106 Find a Disk File

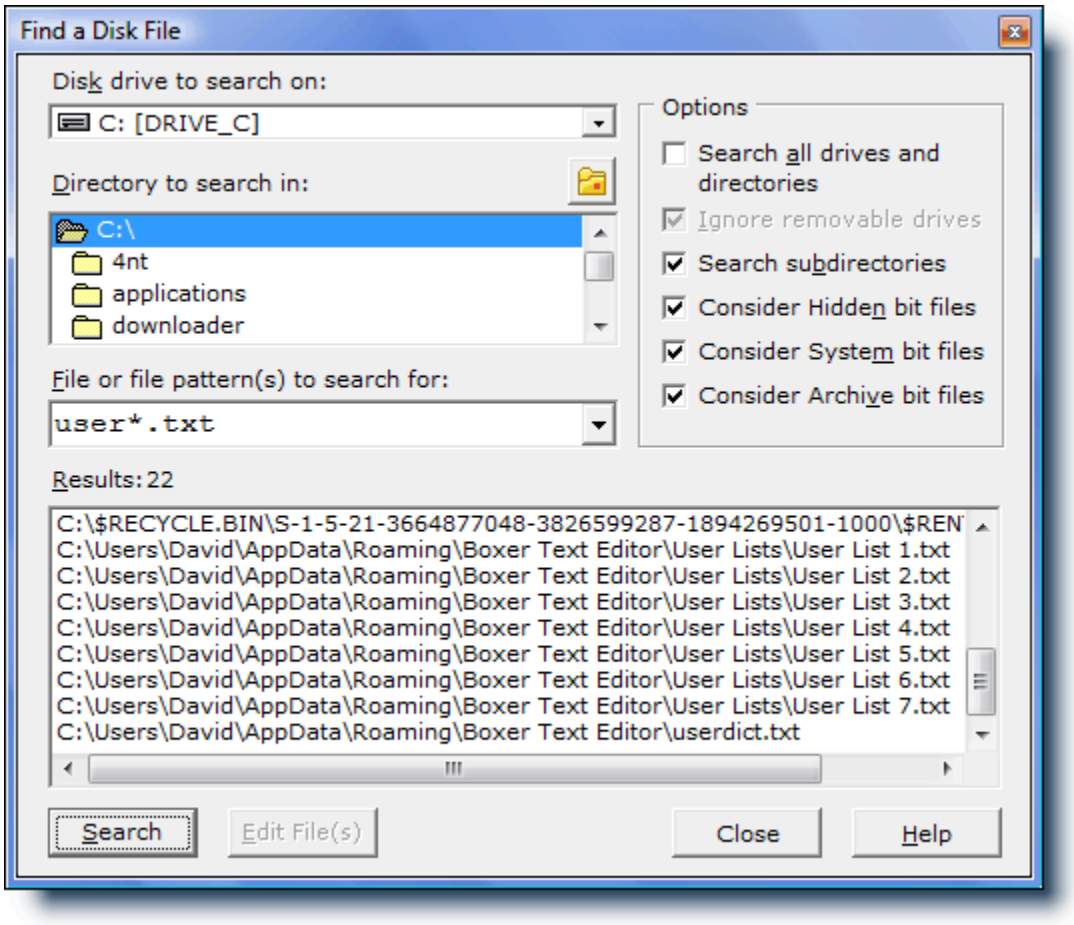
Menu: Search > Find a Disk File

Default Shortcut Key: none

Macro function: FindADiskFile()

The Find a Disk File command provides the ability to locate one or more disk files which match a supplied filename or file pattern(s). Matching filenames are displayed in a results window, and one or more files from the list can then be selected for editing. The dialog box is *non-modal* and stay-on-top, so you can peruse the files opened and later return to the dialog to open other files, without losing the search results.

A range of options are provided to control how the search is conducted, and what drives, directories and files should be considered:



Disk drive to search on
This drop-down list allows you to specify the disk drive to be searched. When the *Search all drives and directories* checkbox is selected, this list is disabled.

Directory to search in
This drop-down list allows you to specify the directory to be searched. When the *Search all drives and directories* checkbox is selected, this list is disabled.


Use the folder icon with the red square in it to jump to the directory of the current file.

File or file pattern(s) to search for
This edit box allows you to specify the filename and/or file pattern(s) to search for. A list of common file patterns has been supplied in the drop-down list, or you can type your own. When specifying multiple patterns, separate the patterns with a semi-colon and do not use intervening spaces.

Options

Search all drives and directories
If this option is selected all drives and subdirectories will be searched, except that removable drives may be exempted using the option below.

Ignore removable drives
If this option is selected drives with removable media (such as floppy drives and mass storage cartridges) will not be searched.

Search subdirectories
If this option is selected all subdirectories below the selected directory will also be searched.

Consider Hidden bit files
If this option is selected, files whose Hidden attribute bit is set will be considered during the file search. The Hidden attribute causes a file to become invisible to many directory listing programs, and is often used together with the System file attribute.

Consider System bit files
If this option is selected, files whose System attribute bit is set will be considered during the file search. The System attribute is sometimes used by the operating system to distinguish files which should not be altered or deleted.

Consider Archive bit files

If this option is selected, files whose Archive attribute bit is set will be considered during the file search. The Archive attribute is used by the operating system to flag those files which have been changed since the previous backup operation. Backup programs will typically reset a file's Archive bit after saving the file to a backup device. In most cases you will want to leave this checkbox active to ensure that recently changed files will be searched.

5.107 Find and Count

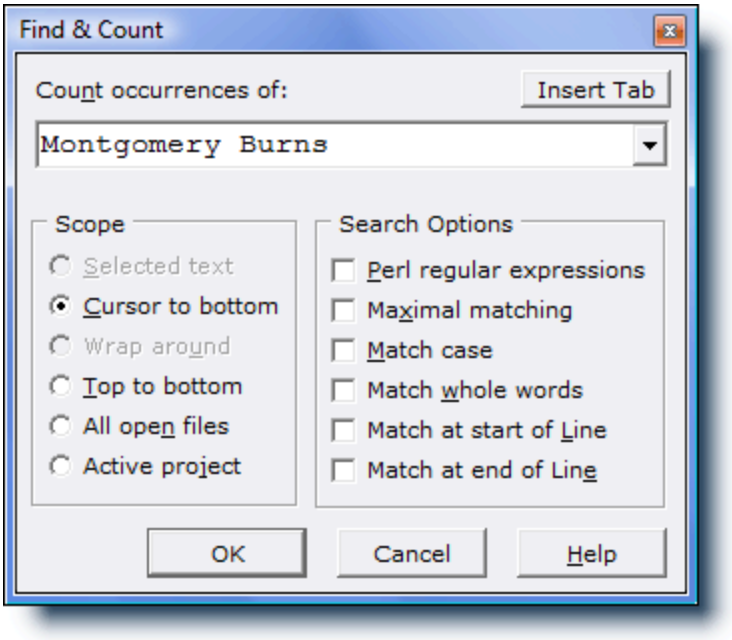
Menu: Search > Find and Count

Default Shortcut Key: none

Macro function: FindAndCount()

The Find and Count command can be used to count the number of occurrences of a specified text string within the current file, or within all edited files. Find and Count is a passive operation, it simply reports the number of matches found for the specified string, within the specified range.

The search options that appear in the Find and Count dialog box are the same as those which appear in the Replace dialog box. See the Replace topic for full details.



The Find and Count command reports its result using a read-only edit box so that the value can be copied to the Windows clipboard.

5.108 Find Differing Lines

Menu: Search > Find Differing Lines

Default Shortcut Key: Ctrl+D

Macro function: FindDifferingLines()

The Find Differing Lines command can be used to locate differing lines among two or more similar files. After the command is issued, the text cursor will be advanced in

each open file to the next line whose text is not identical in among all open files.

This command will be most efficient when used as follows: open the files to be compared and select Window | Tile Across or Window | Tile Down to arrange the windows in a left-to-right or top-to-bottom arrangement. Position the text cursor in each file to line 1, or to a line just before where the comparison is to begin. *In any case, the text on each starting line should be identical among all the files being compared*. When the Find Differing Lines command is issued, the text cursor will be advanced in each file until a line is found which differs among the open files.

The first differing column in the line will be highlighted, and the operation is complete. You can make any corrections that might be needed and then issue the command again to find the next difference. If the difference that was found involves the addition or deletion of one or more lines, the files will need to be re-synchronized manually before proceeding. That is, the text cursor must be moved in each file to a line with identical content so that a new comparison can begin.


Find Differing Lines ignores minimized files during its operation, so if there are any files open which should *not* be compared they can be minimized before beginning.

5.109 Find Distinct Lines

Menu: Search > Find Distinct Lines

Default Shortcut Key: none

Macro function: FindDistinctLines()

The Find Distinct Lines command can be used to isolate all distinct lines within the current file. The distinct lines are copied, with line numbers, into an untitled file. No change is made to the current file during the operation.

If a range of lines is selected, Find Distinct Lines will operate only on that portion of the file.

The results are presented in alphabetic order. The Sort Lines command can be used to sort by line numbers, if desired.

The effect of this command is similar to the Find Unique Lines command, with an important difference: Find Unique Lines omits from its report any lines which are duplicated. Find Distinct Lines includes duplicated lines, but places just a single instance of such lines in its report. An example will clarify:

Original File's Content...	Find Unique Lines gives...	Find Distinct Lines gives...
AAA	AAA	AAA
BBB	DDD	BBB
BBB	EEE	CCC
CCC	FFF	DDD

CCC		EEE
DDD		FFF
EEE		
FFF		

This command can be useful for isolating distinct entries in a list. For example: a file contains lists of email address that were merged from multiple sources. Find Distinct Lines could be used to create a new list that contains one occurrence of each distinct email address. (Following the same example, a similar result could be obtained using the Delete Duplicate Lines command.)

5.110 Find Duplicate lines

Menu: Search > Find Duplicate Lines

Default Shortcut Key: none

Macro function: FindDuplicateLines()

The Find Duplicate Lines command can be used to locate all lines within the current file which are duplicated elsewhere in the file. The duplicate lines are copied, with line numbers, into an untitled file. No change is made to the current file during the operation.

If a range of lines is selected, Find Duplicate Lines will operate only on that portion of the file.

The results are presented in alphabetic order. The Sort Lines command can be used to sort by line numbers, if desired.

If you need to delete duplicate lines, use the Delete Duplicate Lines command.

This command can be useful for finding duplicate items within a list which is expected to contain only unique entries. For example: given a list of charitable donor names, Find Duplicate Lines could be used to find those parties who have contributed more than once.

5.111 Find Fast

Menu: Search > Find Fast

Default Shortcut Key: Ctrl+F3

Macro function: FindFast()

The Find Fast command can be used to quickly search for the next occurrence of the

word beneath the text cursor. The search is performed in the forward direction, toward the end of file. The search options from the Find command dialog box are used, even if the Find command has not yet been used in the current edit session.

5.112 Find Mate

Menu: Search > Find Mate

Default Shortcut Key: Ctrl+]

Macro function: FindMate()

The Find Mate command locates the mating parenthetical element to the parenthetical element at the text cursor, and moves the text cursor ahead (or back) to that position. The search begins at the text cursor and will continue all the way to the start or end of the file, as may be needed.

If the text cursor is sitting on an opening parenthetical character such as `(`, `[`, `<`, or `{`, the cursor will be moved ahead to the corresponding closing mate, with consideration given to nesting. If the cursor is situated on a closing parenthetical character such as `)`, `]`, `>`, or `}`, the cursor will be moved backward to the corresponding opening mate, again with consideration given to nesting.

The Find Mate command also recognizes text strings as parenthetical elements, and many of the most common parenthetical pairs have been pre-defined. For example: if the cursor is sitting on `begin`, Find Mate will locate `end`. If the cursor is sitting on `<i>` (the HTML code to begin italics), Find Mate will find `</i>`. If the cursor is sitting on `while`, Find Mate will find `endwhile`.

The Find Mate command can be used to extend an existing text selection to a closing element. For example, to select a parenthesized block of text, select the opening parenthesis and issue the Find Mate command. The selection will be extended to include all of the text up to and including the closing parenthesis.

The parenthetical pairs recognized by Find Mate can be viewed and/or defined on the Configure | Preferences | Editing 1 options page. The name of the option is *Set mating pairs for Find Mate*.


When editing a file for which syntax highlighting information is available, Find Mate will ignore parenthetical elements that occur within block comments, end-of-line comments, character constants and quoted strings. If syntax highlighting is disabled, or unavailable, this feature cannot be performed.


Find Mate can also be used to test for unmated parenthetical elements, since a request to find a mate for an unbalanced element will result in a report that the closing mate could not be found.


When defining Find Mate pairs for tagged languages such as HTML, remember that commands which include parameters will need different treatment than commands

that cannot use parameters. For example, if you were to define 'table' using the definition `<table>=</table>`, Boxer would not be able to find matches when 'table' was used with parameters, such as `<table width="200">`. For this reason, a definition of the form `<table=</table>` should be used instead, without the closing `>` character.

Find Mate is not able to handle mating pairs whose beginning or ending element is shared by other parenthetical pairs. For example, the definitions `#if=#endif`, `#ifdef=#endif` and `#ifndef=#endif` all share the same closing element, `#endif`. The nesting complexities that could arise from such definitions is beyond the scope of the Find Mate command.

5.113 Find Next

Menu: Search > Find Next

Default Shortcut Key: F3

Macro function: FindNext()

The Find Next command is used to repeat the most recent search in a forward direction. The new search will obey all of the search options which were used when the search was first initiated with the Find command.

When the *Incremental search* and *Select matched text* options are both in use, the Find Next command can be issued from the keyboard in order to display additional matches to the partially typed search string while the Find dialog is still open.

5.114 Find Previous

Menu: Search > Find Previous

Default Shortcut Key: Shift+F3

Macro function: FindPrevious()

The Find Previous command is used to repeat the most recent search in a backward direction. The new search will obey all the of search options which were used when the search was first initiated with the Find command.

When the *Incremental search* and *Select matched text* options are both in use, the Find Previous command can be issued from the keyboard (*Shift+F3*) in order to display additional matches to the partially typed search string while the Find dialog is still open.

5.115 Find Text in Disk Files

Menu: Search > Find Text in Disk Files

Default Shortcut Key: none

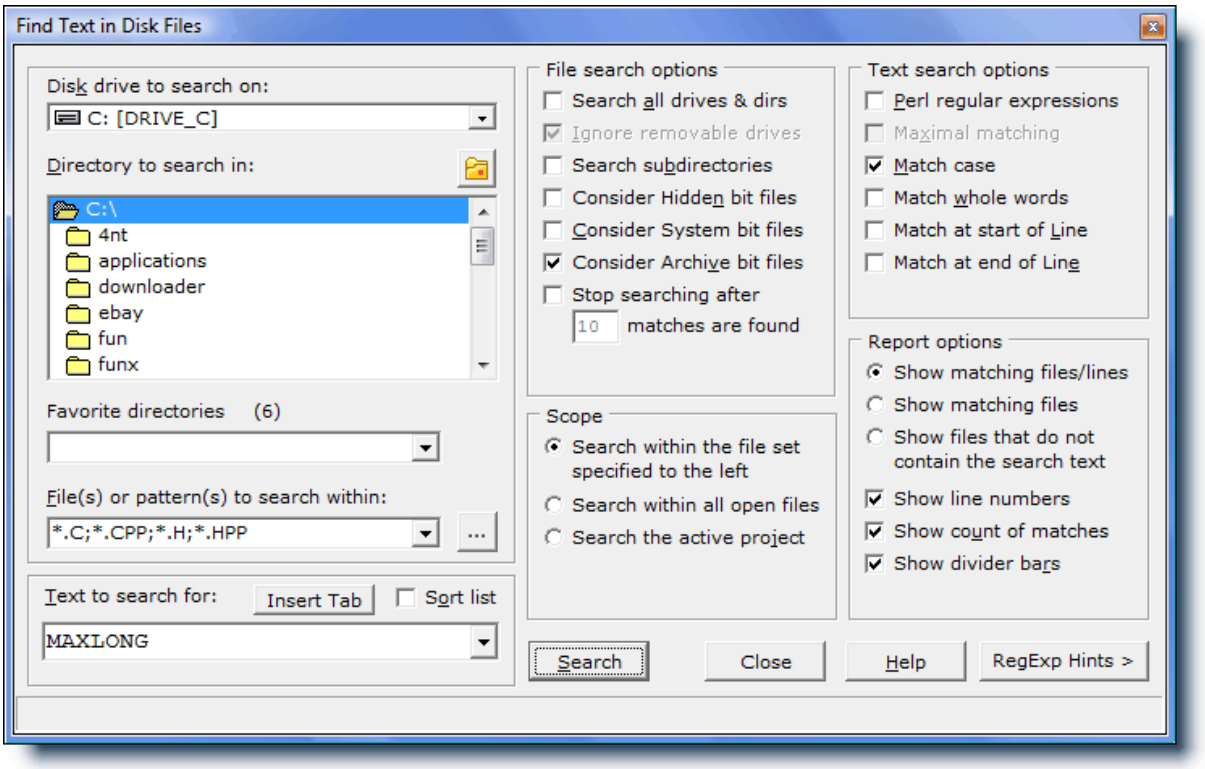
Macro function: FindTextInDiskFiles()

The Find Text in Disk Files command provides the ability to search for a text string across a specified range of drives, directories and files. Lines which contain the desired string are presented in a results window, and the file containing a match can be opened by pressing *Enter* or double clicking on the line.

Regular Expressions can be used when specifying the search string, and one or more file patterns can be used to search within an entire class of files.

The results window is *non-modal*, so you can peruse the files opened and later return to the window to open other files, without losing the search results. The results window has a *Copy All* button which allows its results to be copied to the current clipboard. The *Copy Selected* button will copy only those lines that have been selected. The *Open All* button will automatically open all files in which matching lines were found.

A wide range of options are provided to control how the search is conducted, and what drives, directories and files should be searched:



Disk drive to search on

This drop-down list allows you to specify the disk drive to be searched. When the *Search all drives and directories* checkbox is selected, this list is disabled.

Directory to search in
This drop-down list allows you to specify the directory to be searched. When the *Search all drives and directories* checkbox is selected, this list is disabled. Note that a double-click is required to select a directory; a single-click will not suffice.


Use the folder icon with the red square in it to jump to the directory of the current file.

Favorite directories
This control can be used to recall other directories that have been used in the past.

File(s) or file pattern(s) to search within
This edit box allows you to specify the filename and/or file pattern(s) to search within. A list of common file patterns has been supplied in the drop-down list, or you can type your own. When specifying multiple patterns, separate the patterns with a semi-colon (;) and do not use intervening spaces.


The file patterns that appear in the drop-down list are shared with the File Open dialog. The file patterns which appear in this dialog are user-definable via the Configure | Preferences | File I/O options page.


Regardless of whether or not such files match the supplied filename(s) or file pattern(s), binary files will not be searched by this command.

Text to search for
This edit box is used to specify the text string to be found. Regular Expressions can used if desired. The associated drop-down list can be used to recall previous search strings.


The *Delete* key can be used while the drop-down list is displayed to delete the selected entry from the history list.


Special characters can be entered into the *Text to search for* edit box using the technique described in the Help topic Inserting Special Characters.

Insert Tab
Use this button to insert a tab character into the *Text to search for* edit box.

Ordinarily, the *Tab* key is used to move from field to field within a dialog box. If you would prefer that the Tab key insert a tab character in this dialog box, and in other Find/Replace related dialog boxes, check the relevant box on the Configure | Preferences | Tabs dialog page.

Sort List
If this box is checked the history list will be maintained in alphabetic order, rather than in the order the strings were entered.

When switching to an alphabetically sorted list, the chronological ordering of the list will be lost, and cannot be restored by unchecking the checkbox.

File Search Options

Search all drives and directories
If this option is selected all drives and subdirectories will be searched, except that removable drives may be exempted using the option below.

Ignore removable drives
If this option is selected drives with removable media (such as floppy drives and mass storage cartridges) will not be searched.

Search subdirectories
If this option is selected all subdirectories below the selected directory will also be searched.

Consider Hidden bit files
If this option is selected, files whose Hidden attribute bit is set will be considered during the search. The Hidden attribute causes a file to become invisible to many directory listing programs, and is often used together with the System file attribute.

Consider System bit files
If this option is selected, files whose System attribute bit is set will be considered during the search. The System attribute is sometimes used by the operating system to distinguish files which should not be altered or deleted.

Consider Archive bit files
If this option is selected, files whose Archive attribute bit is set will be considered during the search. The Archive attribute is used by the operating system to flag those files which have been changed since the previous backup operation. Backup programs will typically reset a file's Archive bit after saving the file to a backup device. In most cases you will want to leave this checkbox active to ensure that recently changed files will be searched.

Stop searching after *n* matches are found
Use this option to stop the search after a specified number of matches have been found.

Scope

Search within the file set specified to the left
Use this option to search a file set which has been designated in the disk, directory and file controls at the left side of the dialog.

Search within all open files
Use this option to restrict the search to those files that are currently open for editing.

Search the active project
Use this option to limit the scope of the Find operation to those files within the active project.

Text Search options

Perl regular expressions
If this box is checked, wildcard characters within the search string will be interpreted according to the Perl-Compatible Regular Expression (PCRE) convention. In part, this means that the asterisk (*) will cause a match of zero or more occurrences of the preceding character. The period (.) will match any single character. For more information, see Regular Expressions.

Maximal matching
When using pattern matching characters, there can sometimes be more than one text string that matches the search string. This option can be used to request that the longest possible matching string be returned.

Match case
This option can be used to force the search string to be matched exactly. When unchecked, a case insensitive search is performed.

Match whole words
This option can be used to restrict matches to those strings which appear as a whole word. The characters which serve to delimit words are user-configurable; see Configure | Preferences | Cursor.

Match at start of line
This option can be used to force the search string to be matched only when a matching string appears at the start of a line. This effect can also be achieved with a Regular Expression.

Match at end of line
This option can be used to force the search string to be matched only when a matching string appears at the end of a line. This effect can also be achieved with a Regular Expression.

Report Options

Show matching files/lines
If this option is selected, the text of the matching lines will be shown in the results window, grouped by file.

Show matching files
If this option is selected, the filenames in which matching lines occurred will be reported, but the matching lines will not be shown..

Show files that do not contain the search text
If this option is selected, the filenames in which matching lines do *not* appear will be reported..

Show line numbers
If this option is selected, line numbers will be displayed to the left of each matching line.

Show count of matches
If this option is selected, the count of matches found in each file will be shown in the results window.

Show divider bars

If this option is selected, divider bars will be used within the results window to separate one file's matches from another.


If you prefer that the Find Text in Disk Files dialog automatically start in the directory of the current file, use the relevant option on the Configure | Preferences | File I/O dialog page.

5.116 Find Unique Lines

Menu: Search > Find Unique Lines

Default Shortcut Key: none

Macro function: FindUniqueLines()

The Find Unique Lines command can be used to locate all lines within the current file which are *not* duplicated elsewhere in the file. The unique lines are copied, with line numbers, into an untitled file. No change is made to the current file during the operation.

If a range of lines is selected, Find Unique Lines will operate only on that portion of the file.

The results are presented in alphabetic order. The Sort Lines command can be used to sort by line numbers, if desired.

The effect of this command is similar to the Find Distinct Lines command, with an important difference: Find Unique Lines omits any lines which are duplicated from its report. Find Distinct Lines includes duplicated lines, but places just a single instance of such lines in its report. An example will clarify:

Original File's Content...	Find Unique Lines gives...	Find Distinct Lines gives...
AAA	AAA	AAA
BBB	DDD	BBB
BBB	EEE	CCC
CCC	FFF	DDD
CCC		EEE
DDD		FFF

EEE		
FFF		

This command can be useful for finding items within a list which are not duplicated elsewhere in the list. For example: given a list of zip codes to which deliveries must be made, Find Unique Lines could be used to find those zip codes for which only one delivery must be made, allowing special arrangements to be made.

5.117 Flip Case

Menu: Edit > Flip Case

Default Shortcut Key: Shift+Ctrl+F

Macro function: FlipCase()

The Flip Case command flips (toggles, inverts) the case of the character at the text cursor, and moves the cursor to the next character in the line. Use this command to quickly convert a short string of lowercase characters to uppercase, or uppercase characters to lowercase, by issuing the command repeatedly to move through the string.

5.118 Format XML / XHTML

Menu: Tools > Format XML / XHTML

Default Shortcut Key: none

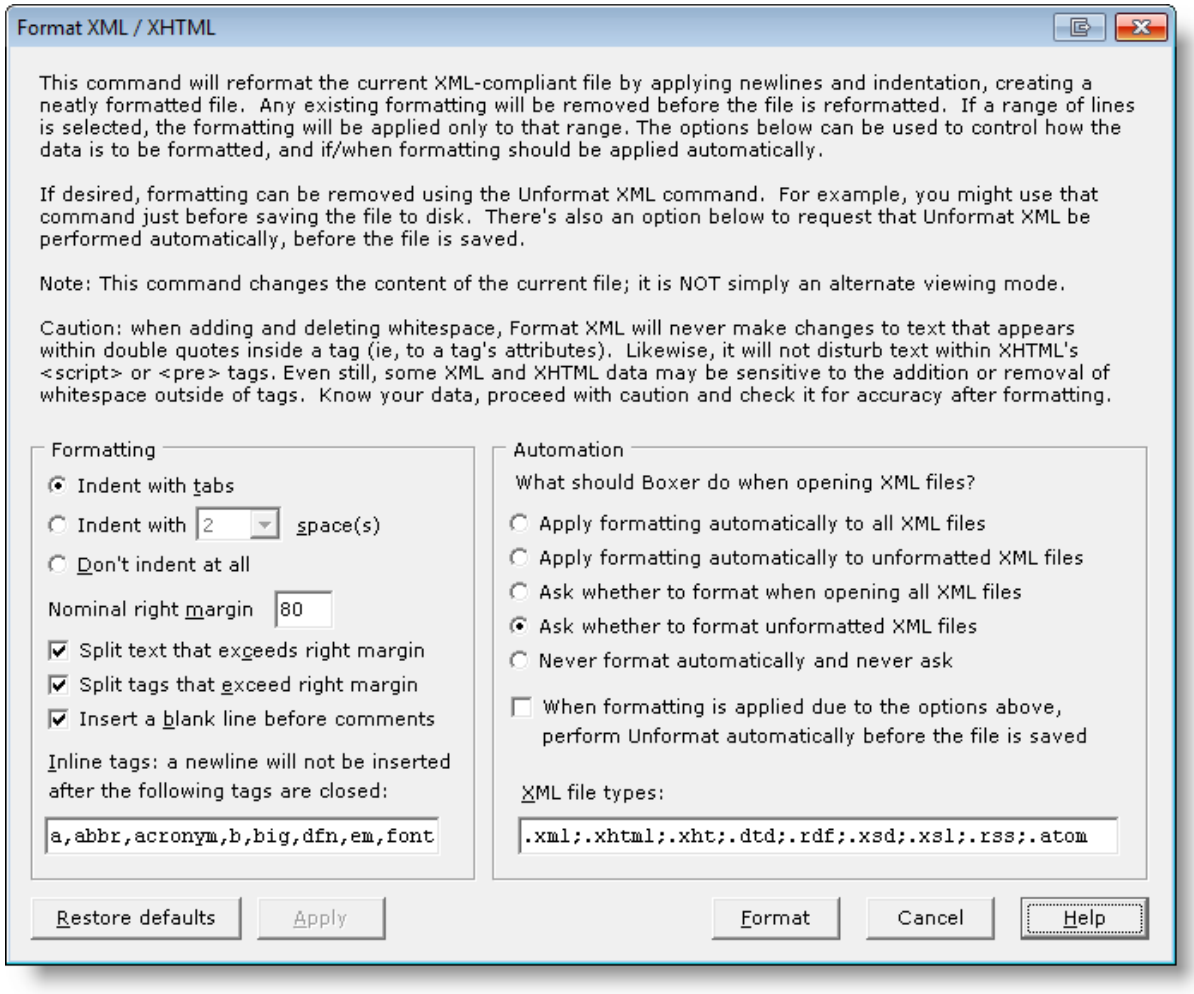
Macro function: FormatXML()

The Format XML / XHTML command can be used to apply formatting to XML-compliant files. If you've ever worked with XML files, you may have noticed that these files often lack line enders and indenting. When these files are opened in Boxer, the text appears as a single long line, and flows off-screen and out of sight at the right edge of the window. The absence of formatting presumably provides some efficiency to the software programs that process these files, but does so at the expense of human readability. The Format XML / XHTML command can neatly format these files with line enders and proper indentation.

The Unformat XML / XHTML command can be used to remove formatting.

By default, the formatting operation is applied to the whole file. If a range of lines is selected, formatting will be performed on the range of lines selected.

A variety of options are provided on the Format XML dialog to control the formatting process:



Formatting

Indentation
Three options are provided: indent with tabs, indent with *n* spaces, don't indent at all. If tabs are used, the display value of a tab is governed by the Tab Display Size command. A large indent value can make it easier to see the structure of the data file. But if a document contains deeply nested data blocks, a more modest indent value may need to be used to preserve space.

Right Margin
This option controls the nominal margin at which lines will be split in order to maintain the width of the document. Please note that double-quoted strings will *not* be split in an attempt to stay within the margin. Lines will be split only at legal break points between or within tags.

Split text that exceed right margin
This options controls whether or not text data will be split/wrapped in order to maintain the right margin.

Split tags that exceed right margin
This options controls whether or not tags (with spaces) will be split/wrapped in order to maintain the right margin.

Insert a blank line before comments
This option can be used to force an empty line to appear before a comment line, or a group of comment lines.

Inline tags
This option can be used to list those tags which will be treated as inline tags. When an inline tags is closed, a newline is not added to the output. Inline tags are typically those tags that might be used to apply formatting to a word or phrases, and are not those tags that begin a block of data which will include other tags.

Automation

A variety of options are offered to control when and whether XML formatting should be applied automatically.

Apply formatting automatically to all XML files
If this option is checked, formatting will be applied to all XML files as they are opened, whether formatted or not, without asking.

Apply formatting automatically to unformatted XML files
If this option is checked, formatting will be applied to unformatted XML files as they are opened, without asking. Formatted XML files with be opened without modification.

Ask whether to format when opening all XML files
If this option is checked, Boxer will ask whether formatting should be applied each time an XML file of any kind is opened.

Ask whether to format unformatted XML files
If this option is checked, Boxer will ask whether formatting should be applied each time an unformatted XML file is opened. Formatted XML files with be opened without modification.

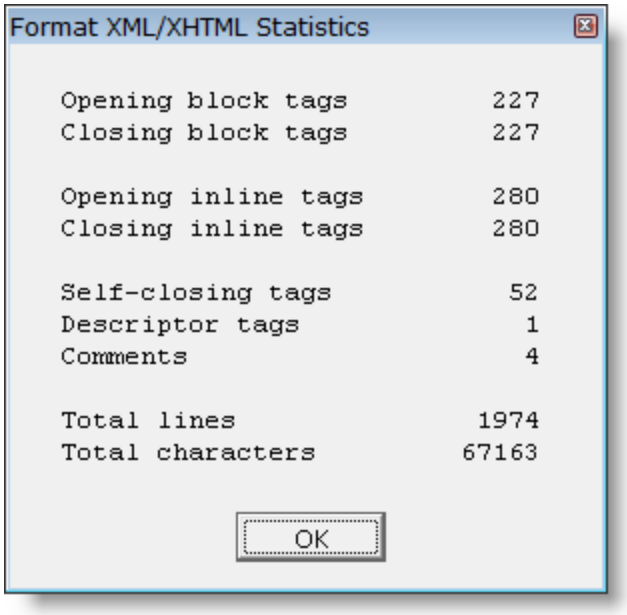
Never format automatically and never ask
If this option is checked, Boxer will never ask about formatting XML files, and will never format them. In this case, the Format XML could be invoked manually in order to initiate formatting.

When formatting is applied due to the options above, perform Unformat automatically before the file is saved
Use this option to ensure that any formatting that is applied due to the automation options is removed before a file is saved. Please note: if the Format XML dialog is summoned manually to add formatting, you'll need to manually remove formatting with Unformat XML, if desired.

XML file types
This option lists the file extensions of those files which are consider XML files, for purpose of automated formatting.

Statistics

When formatting is applied manually (ie, not via automation), a statistics dialog is displayed upon completion:



The dialog can be helpful in locating mismatched or unclosed tags. In particular, the task of converting HTML files to XHTML (XML-compliant HTML) can be aided considerably by using the Format XML command. By using the statistics dialog, and watching for inconsistent indenting, you'll be able to locate which tags are preventing your HTML file from being XML-compliant. A proper XHTML file should start and end at zero indent. If the indent level grows over the course of the document, this is probably an indication that one or more self-closing tags have not been closed. For example, `<br>` needs to be changed to `<br />`.

☞ Text found within `<script>` and `</script>` tags (or within `<?` and `?>`) will not be formatted by this command. Embedded scripting code may be sensitive to indentation and wrapping, so Format XML will not process such text during its operation.

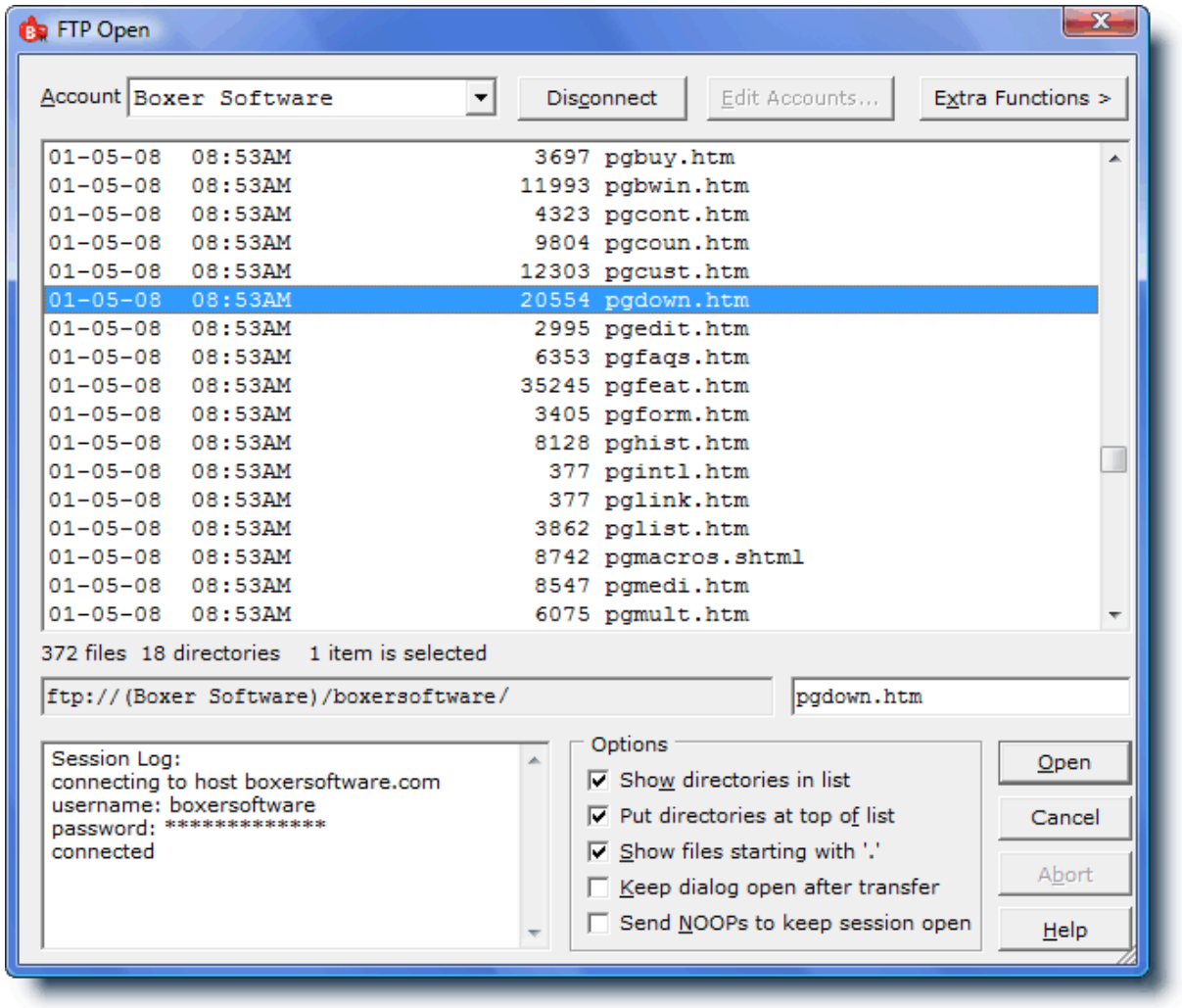
5.119 FTP Open

Menu: File > FTP Open

Default Shortcut Key: Shift+Alt+O

Macro function: none

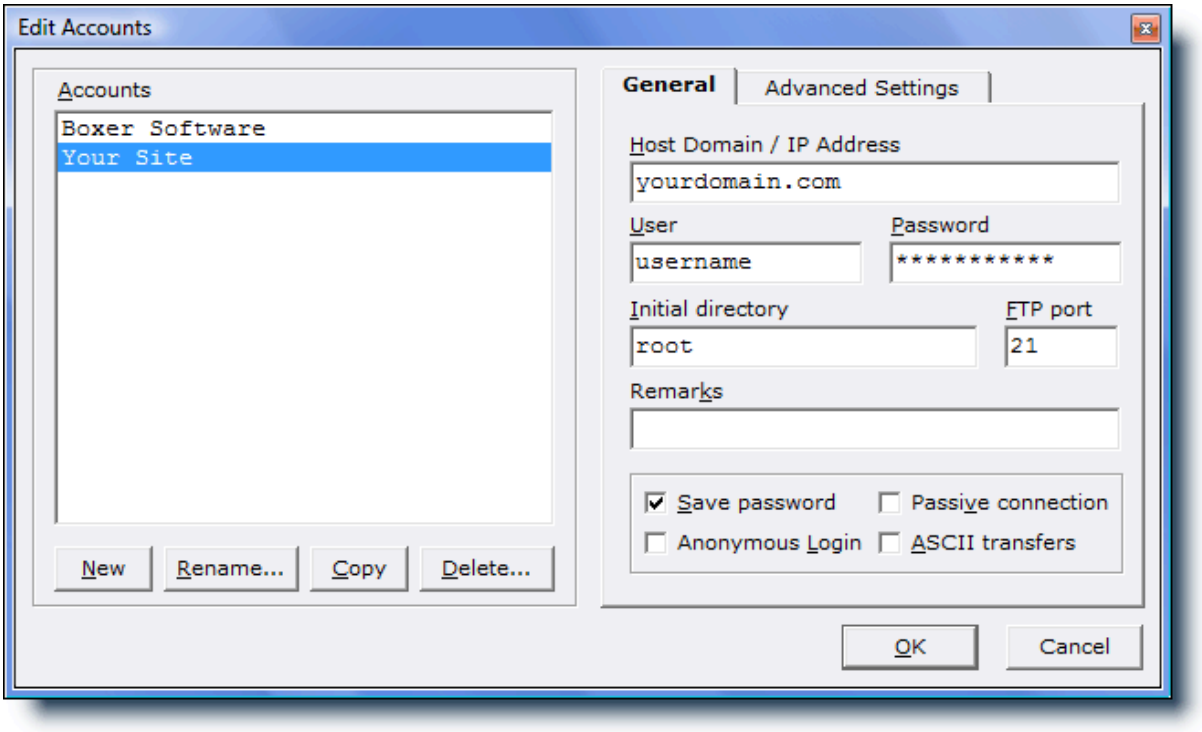
The FTP Open command provides the ability to open files on a remote computer system for viewing and/or editing. A custom dialog is used to display the files on the remote system and to provide access to the various FTP functions:



This help topic is not intended to be a primer on the subject of FTP (File Transfer Protocol) communications. Some familiarity with FTP is assumed on the part of the reader.

Edit Accounts Dialog

The first step to opening files on a remote system is to provide Boxer with the information necessary to establish an FTP connection. The *Edit Accounts* button will present the following dialog:



Initially the dialog will not show any accounts in the *Accounts* listbox at the left. Click *New* to create a new account. Use the *Rename* button to rename the new account as desired.

The *Copy* button can be used to make a copy of the selected account. The *Delete* button can be used to delete a selected account.

Host Domain / IP Address
This field holds the domain name or IP address of the site you wish to connect to.

User
This field holds the user name for the account you have at the remote site. Up to 40 characters are permitted.

Password
This field holds the password for the account you have at the remote site. For security purposes, the field will be displayed with asterisks as it is typed. If the password field is left blank, you will be prompted for the password at connect-time. Up to 40 characters are permitted

☞ If the *Save password* option is selected, the password you enter will be stored in the *Windows registry* along with Boxer's other settings. For security purposes, the password entry will be encrypted in the registry.

Initial Directory
This field holds the name of the directory you would like to be positioned in on the remote system.

If the system you connect to has several directories with different functions, you might choose to create several accounts that differ only in the *Initial Directory* field. This makes it easier to connect directly to the required directory location. The *Copy* button makes it easy to create copies of existing accounts.

FTP port
This field holds the port number to be used for the FTP connection. Unless you have reason to do otherwise, the value should remain at its default setting of 21.

Remark
This field can be used to add a comment about the FTP account being configured.

If you prefer that your password be visible, and security is not an issue, the Remark field can be used to note the password.

Save Password
Use this option to dictate whether or not the *Password* field will be saved from session to session. If this option is *not* selected, the value of the *Password* field will persist only for the current session.

Anonymous Login
Use this option to configure for a system which permits Anonymous login. Selecting this option causes the *User* field to be set to 'anonymous'. The *Password* field is not required, but some systems suggest that your email address be supplied as the password.

Passive Connection
Use this option to request that the FTP session use a passive connection. A passive connection may be required by some firewalls.

ASCII transfers
Use this option to indicate that file transfers are to be performed in ASCII mode. If this option is not selected transfers will be made in Binary mode.

Line ender conversion issues can arise when transferring files from Unix systems. If the results you are seeing are unsatisfactory, try switching the sense of the *ASCII transfers* option.

Advanced Settings
The advanced settings dialog tab is reserved for future use.

FTP Open Dialog

Account
The *Account* list provides a drop-down list of all defined accounts. Select the desired account before clicking *Connect*.

Connect / Disconnect
The *Connect* button is used to initiate an FTP connection. Once the connection is established, the label of this button changes to *Disconnect*.

File List
The *File List* display shows the names of the files on the remote system. The format of this list will depend on the remote system. Boxer does not coerce the data supplied by the remote system into a new format. Likewise, the method used to indicate directory names will vary from system to system. The most common formats use either a 'd' in column one, or the string '<DIR>' somewhere near the filename.


File entries on the remote system which are 'links' or 'redirects' will not be shown in the *File List*. The knowledge of where a link points to is maintained on the remote system. Opening a link via FTP does not have the expected results.

Activity Log
At the lower left of the dialog is the *Activity Log*. This scrolling list maintains a record of the functions that have been performed during the FTP session. The log can be cleared with the *Clear Log* button that appears in the *Extra Functions* panel.

Show directories in list
Use this option to dictate whether or not directory names should appear in the *File List*.

Put directories at top of list
Use this option to dictate whether or not directory names should be placed at the top of the *File List*. If directories are not placed at the top they will appear sorted among the filename entries.

Show files starting with '.'
Use this option to control whether or not filenames that begin with a period should be displayed in the *File List*. On some systems such files are used for configuration and should not be disturbed.

Keep dialog open after transfer
Use this option to control whether the FTP dialog should remain open after the transfer is completed. This may be desirable if multiple files are to be opened from different remote directories.

Send NOOPs to keep session open
Use this option to request that an FTP session be kept open by sending NOOPs (NO Operation commands) to the remote system. This option could be used to defeat a system which enforced an automatic logout after a period of inactivity.


Some systems will not interpret NOOPs as legitimate FTP activity and will proceed with automatic logout.

Extra Functions
The Extra Functions button will toggle on and off a panel that displays additional FTP functions. The following functions are available:

Change To
Use this function to change to the selected directory.


The *Enter* and *Right Arrow* keys can also be used to descend into a selected directory.

Up Dir
Use this function to change to the parent directory.

 The *Left Arrow* can also be used to ascend to a parent directory.

Make Dir
Use this function to create a new directory.

Rename
Use this function to rename a remote file or directory.

Delete
Use this function to delete a remote file or directory. A confirmation will be required before the deletion is performed.

Power Copy
The *Power Copy* function will copy all local files that are open in the editor to the current remote directory. If any of these files have been modified they will be saved automatically. A confirmation is required before the *Power Copy* operation is performed.

 This function can be very useful when editing local copies of website files.

Refresh View
Use this function to refresh the *File List* display. This might be required if you suspect that another program or process has made changes to the remote system that affect the file listing.

Clear Log
Use this function to clear the content of the *Activity Log* window.

Notes

Once a remote file is opened for editing, the name and path of that file will be displayed in the title bar of its edit window. The FTP Filepath includes the information that Boxer needs in order to be able to save the file, or to reopen it at a later time. An FTP Filepath has the form:

```
ftp://(AccountName)/remote_dir/remote_filename.ext
```

Many different areas of Boxer have been enhanced to recognize and process FTP filepaths:

File Save
When the Save command is issued an FTP connection to the remote system will be established automatically so that the file can be saved.

Command Line
When an FTP Filepath appears on Boxer's command line it will automatically be opened for editing when the edit session begins.

Wildcard file specifications are not supported in this context.

Project Files
An FTP Filepath can be placed within a Project file so that it can be opened along with other files named in the Project file.

Most Recently Used Files
When an FTP Filepath is recalled from the Recent Files list near the bottom of the File menu, it will be opened automatically.

Restored Session
When a previous edit session is restored, any remote files that were open in that session will be opened automatically.

Filename at Cursor
If an FTP Filepath is found beneath the text cursor, the Open Filename at Cursor command will open the file automatically.

Spaces are permitted within an FTP account name, but you might wish to avoid using them. Doing so will make it easier to use FTP Filepaths on the command line, or with the Open Filename at Cursor command, since double quoting of the filepath will not be required.

5.120 FTP Save As

Menu: File > FTP Save As

Default Shortcut Key: Shift+Alt+F12

Macro function: none

The FTP Save As command is used to save the current file to a remote computer using a new filename. The file will remain on disk under its old name (except when an untitled file is being saved), and a copy of the file will be saved to the new name and location provided. Boxer will then record the file's new name so that all future save operations are made to the new name.

Boxer's FTP dialog will be used to initiate the connection to the remote computer. For full details about this dialog please see the FTP Open command.

If you are editing a remote file and wish to save a copy locally, use the Save As command.

5.121 Go to Byte Offset

Menu: Jump > Byte Offset

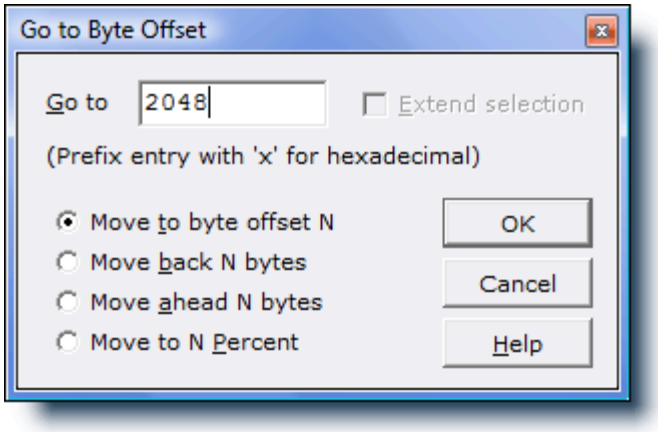
Default Shortcut Key: none

Macro function: GoToByteOffset()

The Go to Offset command can be used to jump immediately to a specified byte offset within the current file. Options are also provided in the Go to Offset dialog box to move backward or forward by the value specified, or to treat the value as a percentage. For example, specifying 50% would result in movement to a character midway through the file.

When viewing a file in Hex Mode, the Go to Byte Offset command will adjust itself to provide the expected movement to positions within the hex display. A *hexadecimal* offset can be specified by prefixing the value entered with an 'x'.

If text is selected when this command is issued, an option will be available to extend the selection to the new location.



The Go To Offset dialog also recognizes the following syntax: +10 to jump ahead 10 bytes; -15 to jump back 15 bytes, and 45% to move to the 45 percent position in the file. The use of this syntax overrides the mode indicated by the radiobutton options.

5.122 Go to Column

Menu: Jump > Go to Column

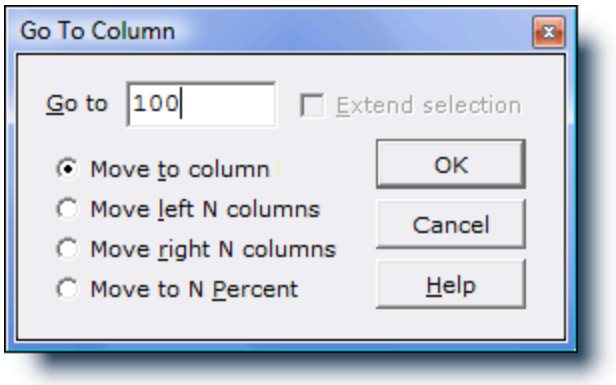
Default Shortcut Key: Shift+Ctrl+G

Macro function: GoToColumn()

The Go to Column command can be used to jump immediately to a specified column number on the current line. Options are also provided in the Go to Column dialog box to move left or right by the value specified, or to treat the value as a percentage. For example, specifying 25% would result in movement to column 25 in a line with 100 characters.

If text is selected when this command is issued, an option will be available to extend

the selection to the new location.



The current column number is always displayed on the Status Bar, next to the 'C' label. The Go to Column command can also be issued by double clicking within the column number display in the Status Bar.

The Go To Column dialog also recognizes the following syntax: +10 to jump ahead 10 columns; -15 to jump back 15 columns, and 45% to move to the 45 percent position along the current line. The use of this syntax overrides the mode indicated by the radiobutton options.

5.123 Go to Line

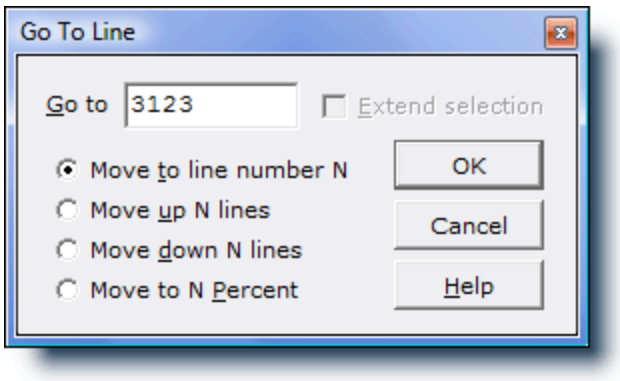
Menu: Jump > Go to Line

Default Shortcut Key: Ctrl+G

Macro function: GoToLine()

The Go to Line command can be used to jump immediately to a specified line number in the current file. Options are also provided in the Go to Line dialog box to move up or down by the value specified, or to treat the value as a percentage. For example, specifying 50% would result in movement to a line midway through the current file.

If text is selected when this command is issued, an option will be available to extend the selection to the new location.



The current line number is always displayed on the Status Bar, next to the 'L' label. The Go to Line command can also be issued by double clicking within the line number display in the Status Bar.

The Go To Line number dialog also recognizes the following syntax: +10 to jump ahead 10 lines; -15 to jump back 15 lines, and 45% to move to the 45 percent position in the file. The use of this syntax overrides the mode indicated by the radiobutton options.

5.124 Go to Paragraph

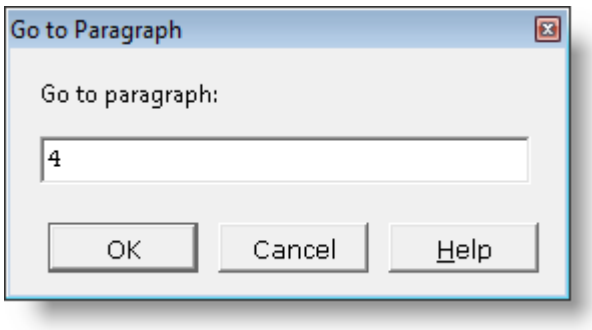
Menu: Jump > Go to Paragraph

Default Shortcut Key: none

Macro function: GoToParagraph()

The Go to Paragraph command can be used to jump immediately to a specified paragraph number in the current file. The distinction between lines and paragraphs relates to the Visual Wrap feature. When Visual Wrap is active, lines with soft line enders are wrapped to width of the window, or to some other wrapping margin. In Visual Wrap mode, a single physical line of text might occupy more than one line on the screen; screen line 11 might correspond to paragraph 4.

Go to Paragraph can be used to move easily among paragraphs:



The distinction between lines and paragraphs will become more obvious if the View

Line Numbers option is active.

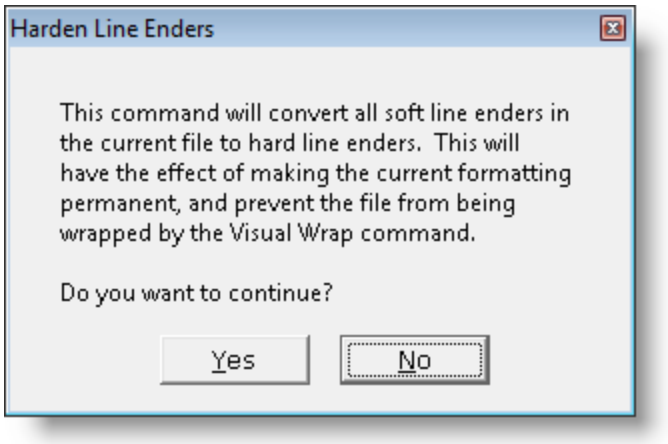
5.125 Harden Line Enders

Menu: Paragraph > Harden Line Enders

Default Shortcut Key: none

Macro function: HardenLineEnders()

The Harden Line Enders command converts soft line enders to hard line enders. If a selection is present, the operation is restricted to the selected range of lines. If a selection is not present, the operation is performed across the whole file. A confirmation dialog will appear before the operation is performed:



The concept of "soft" and "hard" line enders relates to the Visual Wrap command. A line with one or more spaces at the end is considered to have a soft line ender. Lines without trailing spaces are considered to have hard line enders. When Visual Wrap mode is active, lines with soft line enders are eligible to be merged with the content of lines below, allowing text to be reformatted to fit within the window width (or whatever other wrapping margin is chosen).

Applying the Harden Line Enders command to a file has the effect of making the current on-screen formatting permanent... until or unless the Soften Line Enders command is used to reverse this operation. If you apply the Harden Line Enders command to a selected range of lines, these lines will be ineligible for wrapping by the Visual Wrap command.

See also: Visual Wrap, Visual Wrap Options, Soften Line Enders

5.126 Help

Menu: Help > Boxer Help

Default Shortcut Key: F1

Help is available at any time within Boxer by pressing F1. In most cases, the help topic presented will be sensitive to the context in which help was requested.

While cursoring within the main menu or a context menu Help will be presented for the highlighted menu item.

Most dialog boxes contain a *Help* button which presents the help topic dealing with that dialog box.

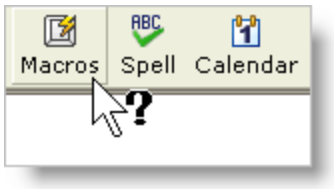
When editing a source code file for which Syntax Highlighting has been defined, the Help command can be used to summon language-specific help information for the word beneath the text cursor. The pathname of the help file which is associated with the language is defined in the *Help File* parameter of the Syntax Highlighting information for that language. See the Syntax Highlighting topic for more information about this capability.

5.127 Help On

Menu: Help > Help On

Default Shortcut Key: Shift+F1

The Help On command is used to activate a special mode in which the mouse cursor is changed to a help icon with an arrow cursor. In this mode the mouse is relieved of its conventional duties, and Help information will be displayed for the next object or menu item clicked upon.



The mouse arrow cursor will change to a 'no' cursor when atop an item for which help is not available, or not applicable.

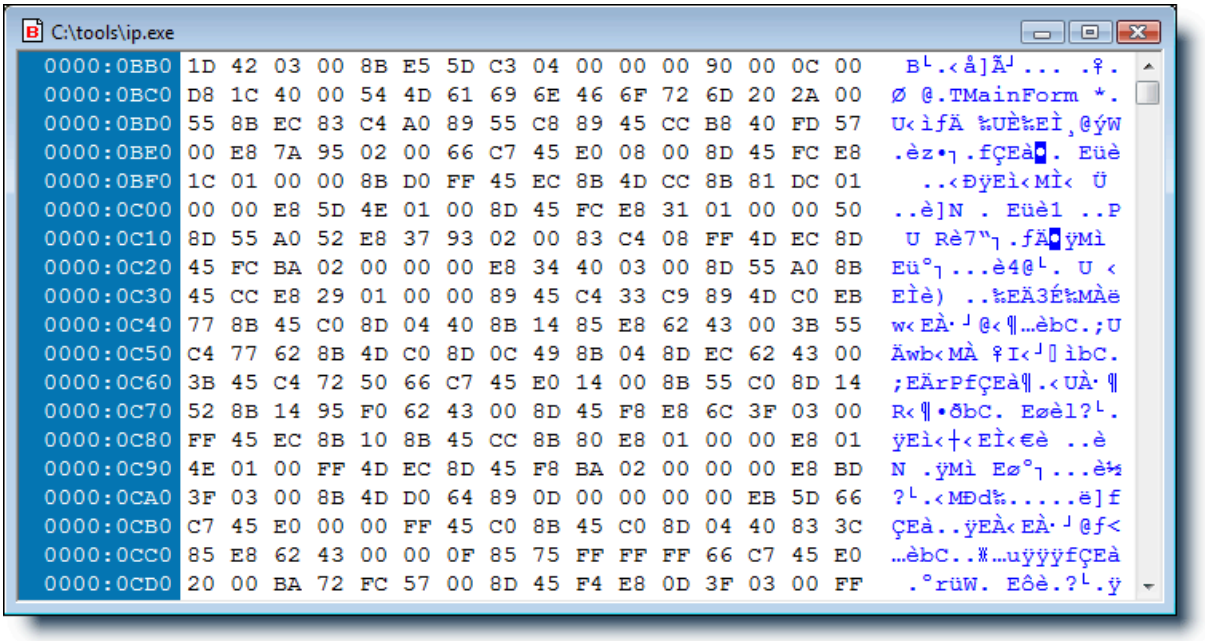
5.128 Hex Mode

Menu: View > Hex Mode

Default Shortcut Key: Shift+Alt+X

Macro function: ViewHexMode()

Use the View Hex Mode command to switch from normal text mode into a read-only hex mode display:



The hex mode display uses a special format which has three sections. At the left, the byte offset into the file is shown in *hexadecimal* format. In the center, sixteen bytes are displayed as two-byte hexadecimal values. At the far right the same sixteen bytes are displayed as characters, except in cases where the character cannot be so represented.

The hex mode display can be exited by issuing this command again, or by pressing *Escape*.

When switching between normal editing mode and hex mode display, the relative location of the text cursor is maintained. This makes the View Hex Mode command useful for studying the hex values characters at or near the text cursor.

The representation of the sixteen characters at the right depends upon whether an ANSI or OEM screen font is in use. The screen font can be changed with the Screen Font command.

Ordinary text files can be opened for editing and then toggled between normal and hex mode display using this command. To open a file for hex mode viewing directly--as is required for the display of binary files--use the Open Hex Mode command instead.

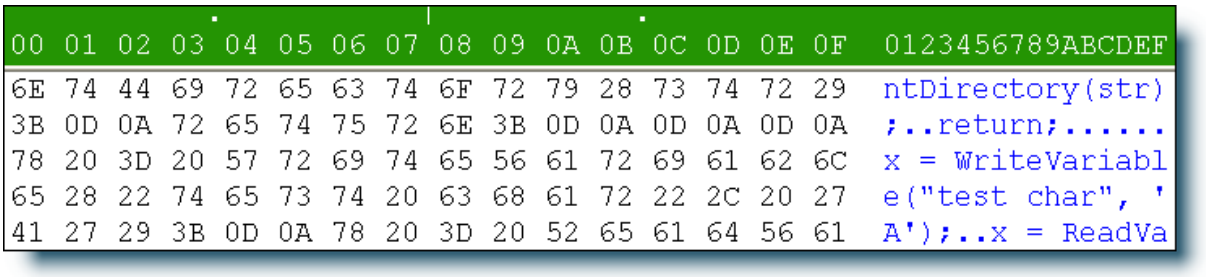
5.129 Hex Ruler

Menu: View > Hex Ruler

Default Shortcut Key: none

Macro function: ViewHexRuler()

The View Hex Ruler command is used to toggle on or off the horizontal ruler at the top of the editing window.



The Hex Ruler labels the column numbers of the file being displayed in hexadecimal format. Clicking on a column number within the ruler will move the text cursor to that column on the current line.

The current byte offset and column number are also displayed in the Status Bar.

To enable the display of *line* numbers, use the View Line Numbers command.

Clicking on the Hex Ruler with the right mouse button provides access to its context menu. The menu has an option to turn off display of the Hex Ruler.

5.130 Horizontal Scroll Bar

Menu: View > Horizontal Scroll Bar

Default Shortcut Key: none

Macro function: ViewHScrollBar()

The View Horizontal Scroll Bar command is used to toggle on or off the scroll bar at the bottom edge on the editing window.

When the current file has no lines which exceed the width of the window, the Horizontal Scroll Bar disappears automatically.

Clicking on the scroll bar with the right mouse button provides access to its context menu. The menu has an option to turn off display of the scroll bar.

5.131 HTML Color Chart

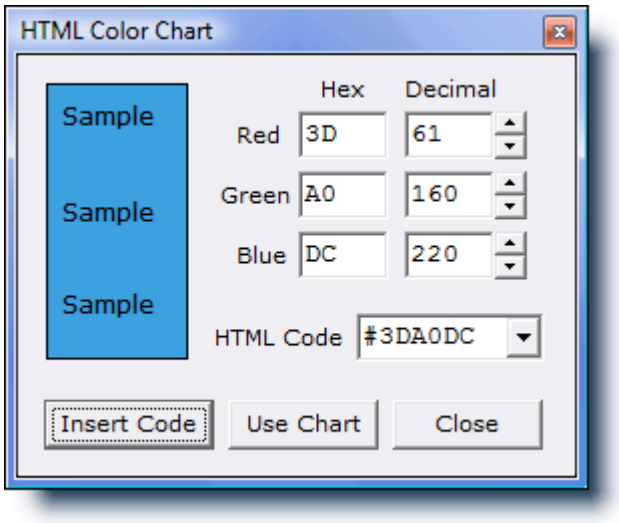
Menu: Tools > HTML Color Chart

Default Shortcut Key: none

Macro function: ColorChart()

The HTML Color Chart command presents a pop-up dialog that allows color values to be

viewed and adjusted by varying the Red, Green and Blue components. RGB values are shown in both *hexadecimal* and *decimal* format..



The current color is displayed in a rectangle at the left, along with sample text in black and white to show the contrast that would result for those color combinations. The value required to display the current color is shown in the box labeled *HTML Code*. The drop-down list holds a history list of recently used color codes. You can type an HTML color value directly into the *HTML Code* combobox, if you wish.

The *Use Chart* button can be used to summon the standard Windows color dialog so that a selection can be made from a color palette. The *Insert Code* button is used to insert the HTML Code into the current text file.

If the HTML Color Chart is left on screen when Boxer is closed, it will be automatically reopened if the edit session is later restored.

5.132 Increment

Menu: Edit > Math > Increment

Default Shortcut Key: none

Macro function: Increment()

The Increment command can be used to increment an integer value (ie, a whole number, not a floating point value) at the text cursor by another integer value. A dialog box will appear to retrieve the value to be added. After clicking 'OK' the arithmetic is performed, and the old value is replaced by the result.

If the text cursor is situated on a character rather than a numeric value, the supplied value will be added to the character at the cursor and the new character will be displayed. If the resultant character value is out of range, an error message will be

given.

5.133 Indent one Space

Menu: Block > Indent One Space

Default Shortcut Key: Ctrl+Space

Macro function: IndentOneSpace()

The Indent One Space command causes a selected range of lines to be indented by one space. On lines which already contain one or more Tab characters of indent, the space character will be applied to the right of the existing indent so that the expected effect is achieved.

If no lines are selected, indentation will be performed on the current line only.


Regardless of the shortcut key assigned to this command, the *Space* key will always perform a block indent when a range of lines is selected. If a small selection is present on a single line the selection will be replaced with a Space character.

5.134 Indent one Tabstop

Menu: Block > Indent One Tabstop

Default Shortcut Key: Shift+Tab

Macro function: IndentOneTabstop()

The Indent one Tabstop command causes a selected range of lines to be indented by one tabstop. Tab options may be set using the Configure | Preferences | Tabs options page.

If no lines are selected, indentation is performed on the current line only.


Pressing the key assigned to the Insert Tab command will also serve to indent a selected range of lines. In most of the keyboard layouts which accompany Boxer the *Tab* key is assigned to the Insert Tab command.

5.135 Indent with String

Menu: Block > Indent with String

Default Shortcut Key: none

Macro function: IndentWithString()

The Indent with String command can be used to simultaneously indent a range of selected lines, and fill the indent region with a user-specified text string.

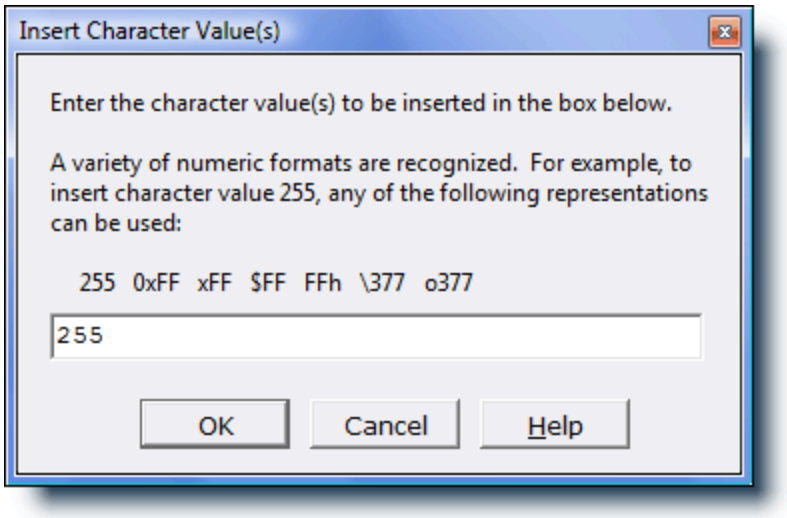
5.136 Insert Character

Menu: Edit > Insert > Character(s)

Default Shortcut Key: none

Macro function: InsertCharacter()

The Insert Character(s) command can be used to insert one or more characters by specifying their numeric values. The values to be entered are typed into a popup dialog box. This command is useful for entering characters which are not readily typed from the keyboard, such as those values below the Space (character value 32), and those above 127.



The ANSI Chart and OEM Chart can also be used to insert non-standard characters into a file. After locating the desired character in the chart, simply press *Enter* or double click on the selected entry.

When the need to insert a special character or symbol arises frequently, consider using the Insert Symbols feature rather than the Insert Character command. The Insert Symbols feature permits a defined character to be entered using a single keystroke.

For additional information, see the Inserting Special Characters topic.

Boxer's Value at Cursor command can be used to verify the value of the character at the cursor.

On most PCs, a character can be entered from the keyboard by typing its numeric value in a special way. With the *Numlock* key on, depress and hold the *Alt* key.

Then type the 0 (zero) on the numeric keypad, followed by the *decimal* value of the character to be inserted. Finally, release the *Alt* key. The character whose value was typed will appear at the text cursor.

5.137 Insert Filename

Menu: Edit > Insert > Filename

Default Shortcut Key: Tab

Macro function: InsertFilename()

The Insert Filename command inserts the full filepath of the current window into the edited text.

To copy the filepath of the current window to the clipboard, use the Copy Filename command.


When editing source code, use this command to quickly place the name of the file into a program comment.

5.138 Insert Formfeed

Menu: Edit > Insert > Formfeed

Default Shortcut Key: none

Macro function: Formfeed()

The Insert Formfeed command can be used to quickly insert the formfeed, character value 12, at the current text cursor location. The formfeed character is recognized by printers as a request to eject the current page and advance to (or load) a new page.

When printing a text file from within Boxer, a formfeed character can be placed in column one--or in the last position on a line--to indicate that a new page should begin at that point. The footer of the page--if one has been defined--will be printed and the page will eject. A formfeed in any other column will be ignored by Boxer's printing service.

5.139 Insert HTML Image Tag

Menu: Edit > Insert > HTML Image Tag

Default Shortcut Key: none

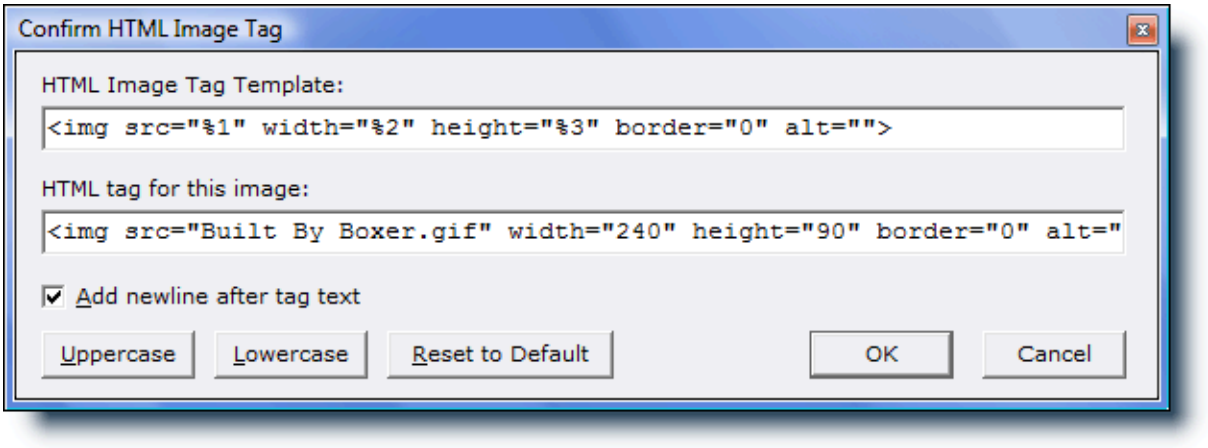
Macro function: HTMLImageTag()

The Insert HTML Image Tag command can be used to insert an HTML image tag into the current file for a selected graphics file. The image tag will use the filename, image

height, and image width of the selected image file. The following image file formats are supported: BMP, GIF and JPEG.

A dialog will appear so that the name of the image file can be selected. If you like, multiple image files can be selected at the same time.

Before the image tag is created, a dialog appears to confirm the operation, and to provide access to the image template:



You can control the format of the image tag by editing the *HTML Image Tag Template* in the upper edit box. The format of the template string can be changed freely, so long as the `%1`, `%2` and `%3` sequences appear in the string, and remain associated with the filename (`src`), `width` and `height` properties, respectively. The image tag that will be inserted appears in the lower edit box. Buttons are provided to quickly convert the tag to uppercase or lowercase, as well as a *Reset to Default* button that will restore the template string to its original form.

You can also activate the Insert HTML Image Tag feature by dragging and dropping images onto Boxer.

5.140 Insert Line Above

Menu: Edit > Insert > Line Above

Default Shortcut Key: Shift+Ctrl+Enter

Macro function: InsertLineAbove()

The Insert Line Above command can be used to create a new line above the current line. The effect of this command is the same as moving the cursor to the end of the previous line and pressing *Enter*, while in Insert mode.

5.141 Insert Line Below

Menu: Edit > Insert > Line Below

Default Shortcut Key: Ctrl+Enter

Macro function: InsertLineBelow()

The Insert Line Below command can be used to create a new line beneath the current line. The effect of this command is the same as moving the cursor to the end of line and pressing *Enter* while in Insert mode.

5.142 Insert Long Date

Menu: Edit > Insert > Long Date

Default Shortcut Key: Shift+Ctrl+F11

Macro function: InsertLongDate()

The Insert Long Date command can be used to insert a text string representing the current date, in *long date* format. A preview of the string which will be inserted is displayed on the Status Bar when the menu cursor is moved onto the Long Date menu entry.

The format used to display the long date is in accordance with the regional settings for date display as defined on your computer. To change these settings, see Start Menu | Settings | Control Panel | Regional Settings | Date.

5.143 Insert Long Time

Menu: Edit > Insert > Long Time

Default Shortcut Key: Shift+Ctrl+F12

Macro function: InsertLongTime()

The Insert Long Time command can be used to insert a text string representing the current time, in *long time* format. A preview of the string which will be inserted is displayed on the Status Bar when the menu cursor is moved onto the Long Time menu entry.

The format used to display the long time is in accordance with the regional settings for time display as defined on your computer. To change these settings, see Start Menu | Settings | Control Panel | Regional Settings | Time.

5.144 Insert Short Date

Menu: Edit > Insert > Short Date

Default Shortcut Key: Shift+F11

Macro function: InsertShortDate()

The Insert Short Date command can be used to insert a text string representing the current date, in *short date* format. A preview of the string which will be inserted is displayed on the Status Bar when the menu cursor is moved onto the Short Date menu entry.

The format used to display the short date is in accordance with the regional settings for date display as defined on your computer. To change these settings, see Start Menu | Settings | Control Panel | Regional Settings | Date.

Double clicking atop the date display on the Status Bar will also issue the Insert Short Date command.

5.145 Insert Short Time

Menu: Edit > Insert > Short Time

Default Shortcut Key: Shift+F12

Macro function: InsertShortTime()

The Insert Short Time command can be used to insert a text string representing the current time, in *short time* format. A preview of the string which will be inserted is displayed on the Status Bar when the menu cursor is moved onto the Short Time menu entry.

The format used to display the short time is in accordance with the regional settings for time display as defined on your computer. To change these settings, see Start Menu | Settings | Control Panel | Regional Settings | Time.

Double clicking atop the time display on the Status Bar will also issue the Insert Short Time command.

5.146 Insert Tab

Menu: Edit > Insert > Tab

Default Shortcut Key: Tab

Macro function: Tab()

The Insert Tab command inserts a Tab (character value 9) at the text cursor location. After insertion, the text cursor moves to the next tabstop, as determined by the Tab Display Size.

If the Tab key has been configured to insert Spaces, an equivalent number of Spaces will be inserted instead of a Tab. The option which controls this behavior appears on the Configure | Preferences | Tabs options page. The option is titled *Tab key inserts spaces*.

The Configure | Preferences | Tabs page also contains an option for the Tab key to insert Spaces and obtain its tabstops from the line above. When this option is used, the *Tab* key will advance the text cursor to the next field of data as determined from the line above the current line. This option is especially useful when editing tabular data within a table or chart.

Boxer's default Tab Display Size is 4, which permits program source code with several indent levels to be displayed without exceeding the screen width. Many other programs, and most printers, will treat Tabs as having a display size of 8. You may need to make adjustments in order to print or display files with another program which does not use a Tab display size of 4. One remedy could be to use the Tabs to Spaces command to convert a copy of the file before using it with the other program. Note that Boxer's Print command will automatically convert Tabs to Spaces before sending its data to the printer, so there will be no such difficulty when printing files from within Boxer.

Tabs, Spaces and Newline characters can be made visible with the Visible Spaces command.


If the Insert Tab command is issued when a range of lines is selected, the Indent one Tabstop command will be performed. If a small selection is present on a single line the selection will be replaced with a Tab character.

5.147 Invert Lines

Menu: Block > Invert Lines

Default Shortcut Key: none

Macro function: InvertLines()

The Invert Lines command can be used to invert a range of selected lines. The last selected line will become first, and the first selected line will become the last.

For example, the text:

```
1. Colorado Springs, CO
2. Denver, CO
3. Eugene, OR
4. Las Vegas, NV
5. Los Angeles, CA
```

```
 6. Oakland, CA
 7. Phoenix, AZ
 8. Portland, OR
 9. Pueblo, CO
10. Riverside, CA
```

would become:

```
10. Riverside, CA
 9. Pueblo, CO
 8. Portland, OR
 7. Phoenix, AZ
 6. Oakland, CA
 5. Los Angeles, CA
 4. Las Vegas, NV
 3. Eugene, OR
 2. Denver, CO
 1. Colorado Springs, CO
```

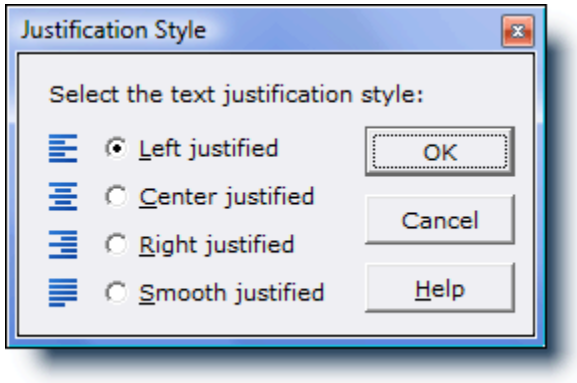
5.148 Justification Style

Menu: Paragraph > Justification Style

Default Shortcut Key: Ctrl+J

Macro function: JustificationStyle()

The Justification Style command is used to set the justification style used by the Reformat, Typing Wrap and Quote and Reformat commands. There are four justification styles to choose from:



The paragraphs below show examples of each justification style:

Left Justified - text will be justified flush against the left edge, with the right edge being left ragged.

Center Justified - text will be centered within

the current text width, with the left and right edges being ragged.

Right Justified - text will be justified flush against the right edge, with the left edge being left ragged.

Smooth Justified - text will be flush against both the left and right margins. Spaces are inserted alternately in the left, center, and right portions of a line to minimize the appearance of 'rivers and valleys' in the justified text.

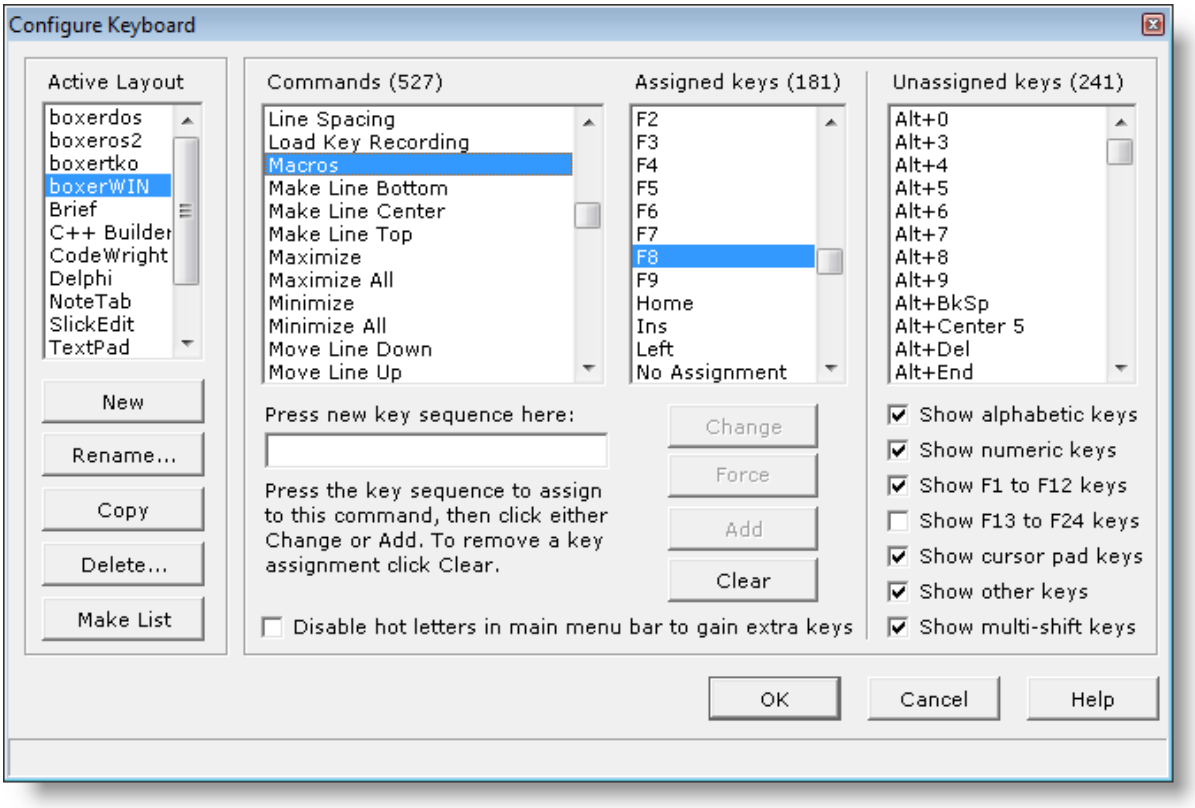
5.149 Keyboard

Menu: Configure > Keyboard

Default Shortcut Key: none

Macro function: ConfigureKeyboard()

The Configure Keyboard command provides the ability to assign key sequences to any of Boxer's commands. With over 450 editor commands, and over 400 key sequences, it's easy to think that keyboard configuration might be a complex undertaking. Not so. Boxer's Configure Keyboard dialog automates the process by providing lists from which *Commands*, *Assigned keys* and *Unassigned keys* can be selected, and by allowing key assignments to be typed directly from the keyboard.



A thorough coverage of the features of the Configure Keyboard dialog box is presented further below. In the paragraphs that immediately follow, a sample configuration session is presented which illustrates how several common changes can be made to the default keyboard layout.

Goal
Create a new keyboard layout which duplicates the default layout, but with a few selected changes.

Discussion
Rather than making changes to a pre-defined keyboard layout, it's always advisable to create a new layout with the *Copy* or *New* button. Future upgrades to Boxer will overwrite the default keyboard layout `BOXERWIN.KBD` and other pre-defined layouts. Custom changes should **not** be made to these files. The *New* button will create a nearly empty layout, but that's not what we want now. Instead, click *Copy* to create a copy of the active keyboard layout, and then click *Rename* to provide the new layout with a name of your choice. You might choose to simply use your first name.

Change One
The Auto-Number command has no key assignment in the default keyboard layout, and you'd like it to have one. Click on the Auto-Number entry in the *Commands* listbox. The *Assigned keys* listbox is updated to show 'No Assignment'. Find a suitable key sequence in the *Unassigned keys* listbox, and click on it. The name of the key sequence selected appears in the edit box beneath the *Commands* listbox. Click the *Change* button to change the assignment for Auto-Number from 'No Assignment' to the

selected key sequence.

Change Two
The Align Right command is assigned to *Ctrl+F9*, but you'd like to use that key for another command instead. Click on the Align Right entry in the *Commands* listbox. Click on the *Clear* button to relieve the command of its key assignment. The *Assigned keys* listbox is updated to show 'No Assignment', and the *Ctrl+F9* key is added to the *Unassigned keys* listbox.

Change Three
The Calculator command is assigned to *F11*, but you'd like it to also be available using the *Ctrl+F9* sequence, which was just freed by Change Two above. Click on the Calculator entry in the *Commands* listbox. Its current assignment of *F11* is displayed in the *Assigned keys* listbox. Click in the edit box beneath the *Commands* listbox to give it focus. Press the *Ctrl+F9* key sequence from the keyboard, and watch its name appear in the edit box. (Whenever you prefer, a key can be pressed in the edit box as an alternative to locating it in the *Unassigned keys* listbox.) Finally, click the *Add* button to create this additional assignment for the Calculator command. A duplicate entry is created for Calculator in the *Commands* listbox, reflecting the fact that there are now two distinct key assignments for this command.

Active Layout listbox

The *Active Layout* listbox displays a list of the available keyboard layouts, and highlights the active layout. Boxer comes with several pre-defined layouts which can make Boxer more closely match the key assignments of another editor or word processor.

If you develop a keyboard layout that matches the key assignments of another popular program, please consider sending it to us at info@boxersoftware.com so that we can make it available to other Boxer users. Keyboard layout files are kept in Boxer's *data folder*, and are given a `.KBD` file extension.

New button
Use the *New* button to create a new keyboard layout. The new layout will contain only the most fundamental key assignments, such as *Up*, *Down*, *Left*, *Right*, etc. The new layout is created with the name 'New'; use the *Rename* button to supply the name of your choice.

Rename button
Use the *Rename* button to change the name of the selected layout to a name of your choice.

Copy button
Use the *Copy* button to make a copy of the active keyboard layout. The new layout will be given the name 'Copy of', prefixed to the name of the active layout. Use the *Rename* button to supply a new name, if desired. Use of the *Copy* button is recommended when you will be making a small number of changes to an existing layout.

Delete button
Use the *Delete* button to delete the selected keyboard layout. A confirmation is required before the layout will be deleted. Once a layout is deleted it **cannot** be recovered, even if *Cancel* is later selected.

Make List button
The *Make List* button creates a file in a new editor window which lists all of the command key assignments in the selected layout. This file could be printed to create a command chart, or saved to disk for later reference.

Commands listbox

The *Command* listbox displays an alphabetical list of all commands which can be reassigned. Clicking on an entry in the *Commands* listbox displays its current assignment in the *Assigned keys* listbox. When the listbox has focus, pressing the first letter of a command will jump the selection bar to that command.

When a command has multiple key assignments, an entry will appear in the *Commands* listbox for each such assignment.

The number of commands displayed in the listbox is shown in parentheses at the top of the list.

Assigned Keys listbox

The *Assigned Keys* listbox displays an alphabetical list of all key sequences which are currently in use. Clicking on an entry in the *Assigned Keys* listbox displays the associated command in the *Commands* listbox. The 'No Assignment' entry does not normally map to a single command, and therefore will not display its associations.

When the listbox has focus, pressing the first letter of a command will jump the selection bar to that command.

The number of key sequences displayed in the listbox is shown in parentheses at the top of the list.

Type new key in this box

This is the edit box where a new key sequence is entered. The edit box can be filled by clicking on an available key sequence from the *Unassigned Keys* listbox, or by pressing a key sequence from the keyboard while the edit box has focus. When a key sequence is entered, its disposition is reported in a message just above the edit box. It might be reported as available, not available, in use, or as being used by the System.

Change button
The *Change* button is used to change the key assignment for the currently selected command to the key sequence displayed in the edit box. The *Change* button will remain disabled until a key which is eligible for assignment has been entered into the edit box.

Force button
The *Force* button is used to change the key assignment for the currently selected command *and* simultaneously remove its assignment from another command.

Add button
The *Add* button is used to create an additional assignment for the selected command. The *Add* button will remain disabled until a key which is eligible for assignment has been entered into the edit box. There is no limit to the number of duplicate key assignments that a command may have.

Clear button
The *Clear* button is used to release a key assignment from the currently selected command. The *Clear* button will be disabled when the current command has no assignment.

Disable hot letters in main menu bar to gain extra keys
This option can be used gain access to the *Alt+letter* key sequences which would otherwise be used to activate the main menu entries. When this option is selected the key sequences *Alt+F*, *Alt+E*, *Alt+B*, etc. become available for assignment to other commands. When the Configure Keyboard dialog is dismissed, the main menu will be redrawn without its hot letters underlined.

Alt+letter sequences which are not otherwise assigned will remain assigned to their respective menus. For example: if this option is selected, but *Alt+F* is not otherwise assigned, it will remain as the key assignment for dropping the File menu.

Regardless of the state of this option, the main menu hot letters will remain functional when the main menu has been activated by tapping the *Alt* key.

When loading a keyboard layout file, Boxer will look for key assignments which conflict with the main menu hot letters in order to determine if this option needs to be checked. If you select this option, but fail to assign any of the *Alt+letter* sequences to other commands, Boxer will sense this when the layout is next loaded, and the option will revert to unchecked. Conversely, if you load a layout which contains one or more key assignments which conflict with the main menu hot letters, Boxer will force this option to checked.

Unassigned Keys listbox

The *Unassigned Keys* listbox displays those key sequences which are available for assignment. Clicking on a key within this listbox causes the key to be displayed in the edit box beneath the *Commands* listbox.

The keys which are to be shown in the listbox can be controlled with various checkboxes:

Show alphabetic keys
Use this option to include the *A-Z* keys, in all their various shift states.

Show numeric keys
Use this option to include the *0-9* keys, in all their various shift states.

Show F1 to F12 keys
Use this option to include the *F1-F12* keys, in all their various shift states.

Show F13 to F24 keys
Use this option to include the *F13-F24* keys, in all their various shift states. Some new keyboards are now offering these additional functional keys.

Show cursor pad keys
Use this option to include the keys from the cursor motion pad, in all their various shift states.

Show other keys
Use this option to include keys which do not group into the categories above.

Show multi-shift keys
Use this option to control whether or not key sequences with multiple shifts should appear in the list.

Notes

- You might notice that the controls in this dialog box do not have hot letters, as do other dialog boxes. This is because the edit box into which key sequences are typed must be able to receive all possible key sequences without losing focus to another control.

- Assigning a key sequence to run a macro is a two-step process. One step is to make the desired assignment to the *Run Macro n* command using the Configure Keyboard dialog. The other step is ensure that the macro itself is numbered accordingly. See the Macros topic for further information.

- It is not possible to use multi-key sequences, such as *Ctrl+K, Ctrl+B* in a key assignment.

- When a given command has more than one key assignment it will have multiple entries in the *Commands* listbox. The first assignment that appears in the *Commands* listbox is called the *primary command assignment*. Additional entries for that command are referred to as *secondary command assignments.* The key sequence associated with the primary command assignment is the one which will be displayed in the main menu next to the command.

- When an *Alt+Letter* sequence is used as a secondary command assignment, you will likely notice a beep when that key sequence is pressed. The beep occurs because the *Alt+Letter* sequence does not map to an underlined hot letter on the main menu bar, and it does not appear as a *shortcut key* in any of the main menu entries. The beep can be silenced by making the *Alt+Letter* sequence the primary command assignment, and letting the existing assignment become the secondary assignment. To do this, the existing primary assignment must be cleared, so the *Alt+Letter* assignment becomes the primary assignment. Then, the other assignment can be added back as the secondary assignment. The only effective difference between a

primary assignment and a secondary assignment is that the primary assignment is displayed in the main menu.


Because they appear in the main menu, primary command assignments are available whenever Boxer is running. Secondary command assignments do not appear in the main menu, and are therefore available only when a child editor window is open. Since most commands are meant to operate on text, this rarely poses a problem. But there are instances where trouble can arise. For example: assume that Ctrl+N is the primary command assignment for the File | New command, and Shift+Alt+N is its secondary assignment. If all child editor windows are closed, the Shift+Alt+N assignment will be non-functional. The primary assignment, Ctrl+N, would need to be used.


The Configure Keyboard dialog recognizes the numeric keypad keys as distinct keys in all of their shifted and unshifted states. These keys appear in the *Unassigned keys* listbox as *Keypad 1*, *Keypad 2*, etc.

5.150 Left Window Edge

Menu: Jump > Left Window Edge

Default Shortcut Key: none

Macro function: LeftWindowEdge()

The Left Window Edge command positions the text cursor to the left edge of the current window. If the file has been scrolled to the right, the amount of scroll will not be affected.


If a text selection is present when this command is issued, the selection will be extended to the new cursor location.

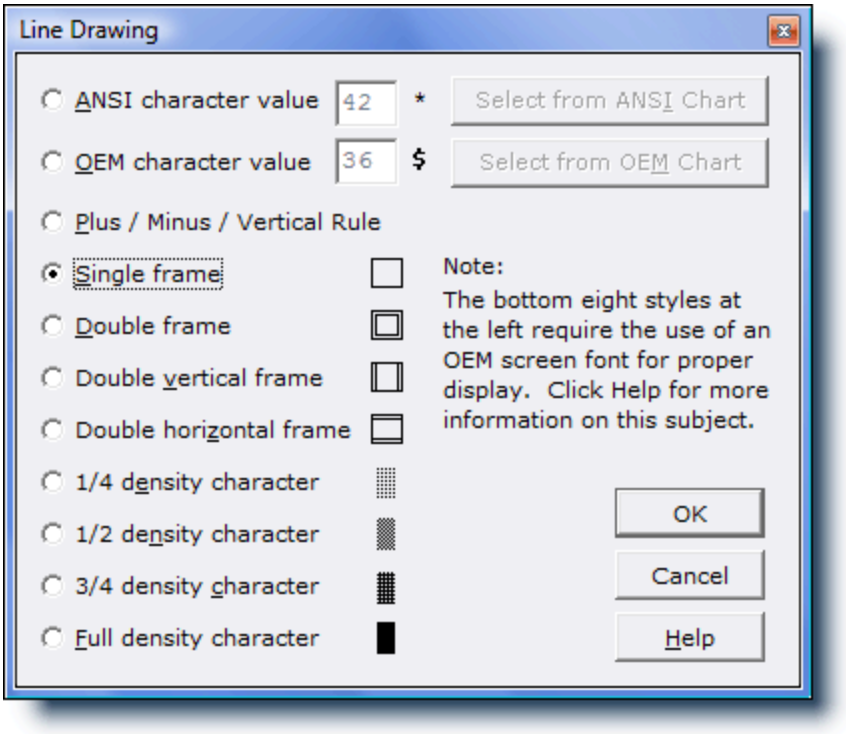
5.151 Line Drawing

Menu: Tools > Line Drawing

Default Shortcut Key: Ctrl+F12

Macro function: LineDrawing()

The Line Drawing command can be used to enter a mode in which the arrow keys are used to draw lines in a selected frame style, or with a specified character. A dialog box appears from which the drawing style is selected:



A drawing style is selected by checking the radio button which corresponds to the desired style. Options are also provided to draw with some other character, and either the ANSI Chart or OEM Chartcan be summoned to assist in character selection.

When using an OEM Screen Font, several additional drawing styles may be used. The bottom eight styles offered in the dialog box require the use of an OEM screen font for proper display. The ANSI character set does not contain these characters, so using these styles with an ANSI Screen Font will produce undesirable results.

Once OK is clicked, Line Drawing mode is active. Use the arrow keys to create lines or boxes as desired. Use *Esc* to cancel Line Drawing mode.

When drawing atop lines which contain Tab characters, the Tabs will be automatically converted to Spaces to ensure proper display. When Line Drawing occurs past the end of a line, the line will be automatically extended with Spaces. If the end of file is encountered, additional blank lines will be added automatically so that Line Drawing can proceed.

When printing files which contain drawing characters from the OEM character set, be sure to use a Printer Font which also uses the OEM character set.

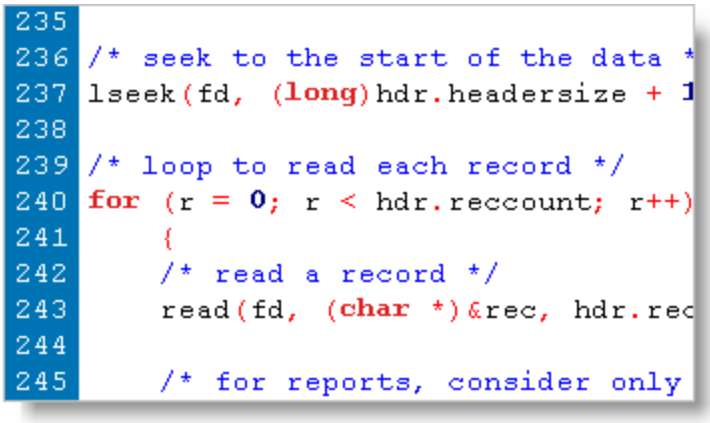
5.152 Line Numbers

Menu: View > Line Numbers

Default Shortcut Key: Alt+F3

Macro function: ViewLineNumbers()

The View Line Numbers command is used to toggle on or off the line numbers in a region to the left of the editing area.



In order to optimize the amount of screen space available for editing, the area allocated to the display of line numbers changes dynamically. If a file grows in size, such that the largest line number requires more space to be displayed, the line number margin will expand automatically. If a file shrinks in size, the line number margin will likewise be adjusted.

When Visual Wrap mode is active, the line number display will be adjusted to display paragraph numbers.

Leading zeros can be displayed on line numbers using an option on the Configure | Preferences | Display options page. The option is titled *Display leading zeros on line numbers*.

The display of Line Numbers is a visual aid and does not result in any changes to the file being edited. To insert line numbers into the file itself, use the Auto-Number command.

Clicking in the Line Number region with the right mouse button provides access to its context menu. Options are available to toggle on and off the display of leading zeros, and to turn off the viewing of line numbers.

The current line number is also displayed in the Status Bar.

To enable the display of a ruler which labels screen *columns*, use the View Column Ruler command.

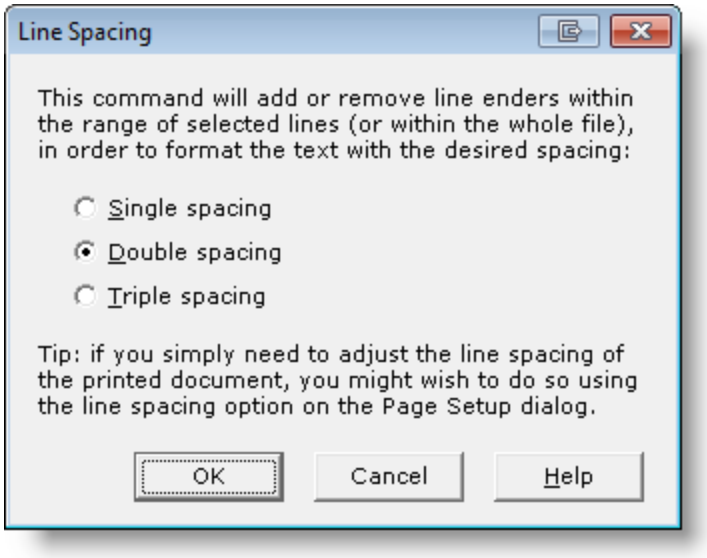
5.153 Line Spacing

Menu: Block > Line Spacing

Default Shortcut Key: none

Macro function: LineSpacing()

The Line Spacing command allows a range of selected lines to be converted to single-, double- or triple-spaced format. If no selection is present, the operation will be performed across the entire file.



If you simply need to adjust the line spacing of the printed document, you might wish to do so using the line spacing option on the Page Setup dialog.

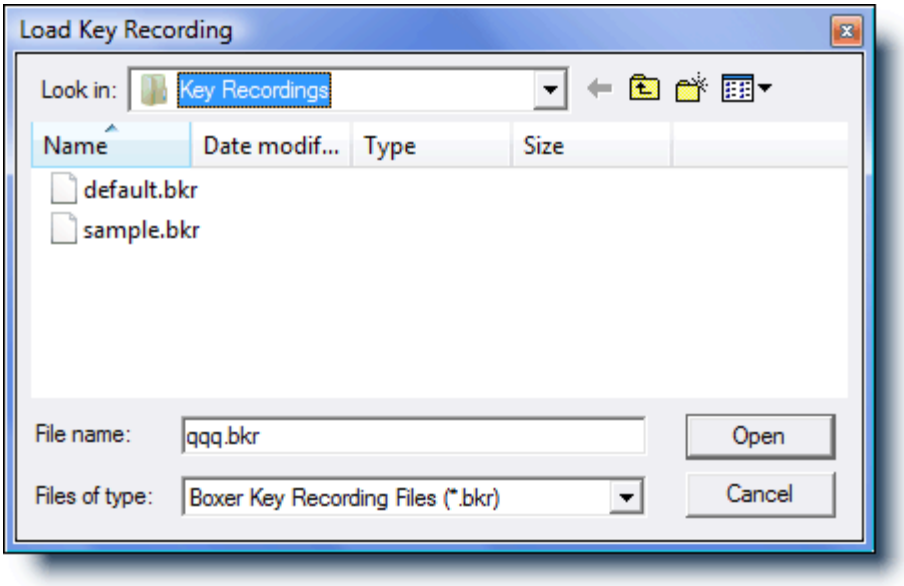
5.154 Load Key Recording

Menu: Tools > Load Key Recording

Default Shortcut Key: none

Macro function: none

The Load Key Recording command can be used to load an existing key recording from disk. A dialog will appear that allows a name to be selected. By default, key recordings are loaded from Boxer's 'Key Recordings' subdirectory.



Once a key recording has been loaded, use the Playback Keys command to playback the recording.

5.155 Macros

Menu: Tools > Macros

Default Shortcut Key: F8

Macro function: none

Boxer includes a powerful macro language than can be used to automate repetitive editing tasks, or to perform specialized processing on the text files you edit. Macros can be created in one of two ways: Macros can be recorded 'by example' by typing commands and/or insertable text within the macro dialog. When this is done, the macro code is written automatically, on-the-fly, in the editor window of the macro dialog. For more complex macros, the edit window can be used to write a macro by hand, or to make refinements to a macro that was recorded by example.

Boxer's macro language is similar in style to the C programming language, and will be quickly understood by anyone who has programmed in a high-level language, or in other macro/scripting languages. The macro dialog contains built-in lists of all the language's keywords, functions and operators, along with instant help information for each entry (see screen shots below). Boxer has been supplied with numerous example macros which are meant to illustrate the use of the language, as well as to provide genuinely useful services. For example, the `ExampleApplyHTML` macro will apply the necessary HTML declarations to make a simple text file into an HTML document.

Some people will want to dive right in, so here's a quick example:

Simple Macro Example, Step One

You've got a file that needs some repetitive editing. You need to delete the first four characters from the start of every third line. The file to be processed is open for editing, and the text cursor is sitting on the first line that needs adjustment. Here's how to create a macro to perform the editing required:

```
Issue the Tools|Macros command from the Main Menu
Click New
Press the Delete key four times
Press the Down Arrow key three times
Click Save
Enter a name for the macro
Click Run, as required
```

The resulting macro looks like this:

```
macro newmacro()
{
Delete;
Delete;
Delete;
Delete;
Down;
Down;
Down;
}
```

Simple Macro Example, Step Two

That's great, you say, but maybe your file is 300 lines long. Or 30,000 lines long. How can we make this macro work on the whole file?

In Step One, the macro was written for you automatically, as you typed the editing commands. To handle a file of arbitrary length, we'll need to add a little code. Edit your macro to look like this:

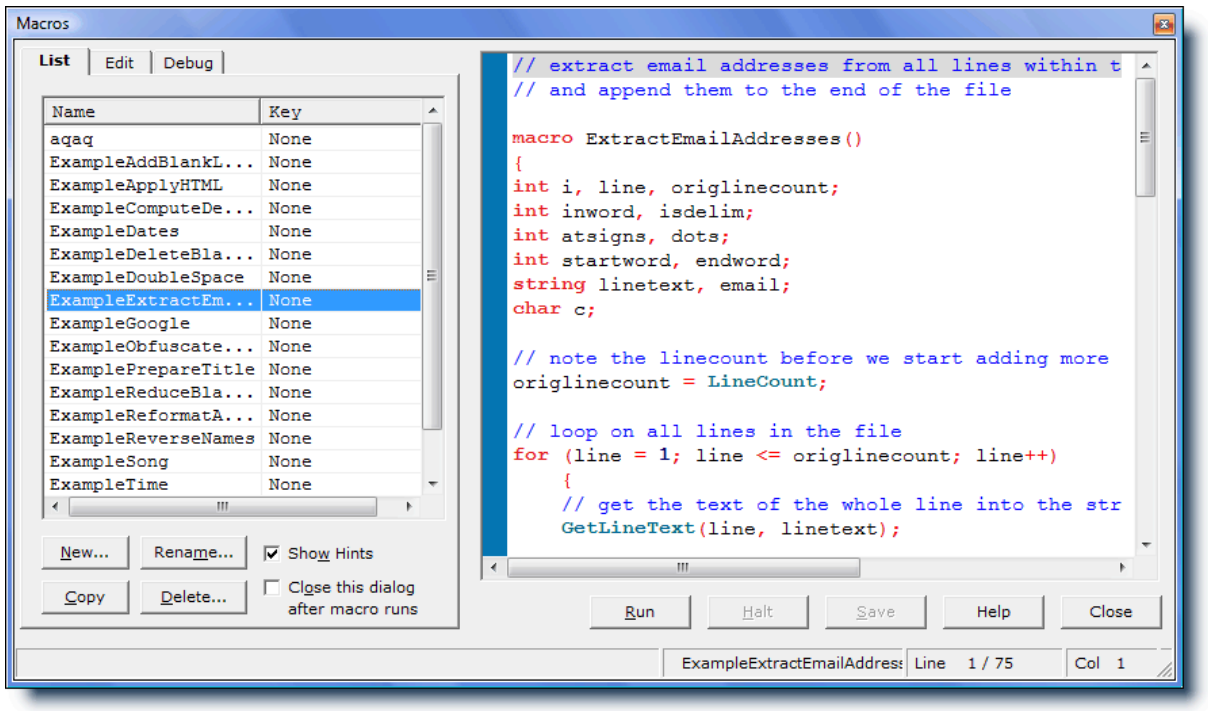
```
macro newmacro()
{
int i;

for (i = 1; i <= LineCount(); i += 3)
    {
    Delete;
    Delete;
    Delete;
    Delete;
    Down;
    Down;
    Down;
    }
}
```

Click *Save*, and then *Run*. This macro loops through the file, counting by three,

performing the necessary adjustments. Because it calls the function `LineCount()`, it will work for a file of any size.

See the following help topics for additional information about macros: Macro Language Reference, Macro Function Reference and Macro Examples.

The sections below cover the Macro Dialog in further detail...



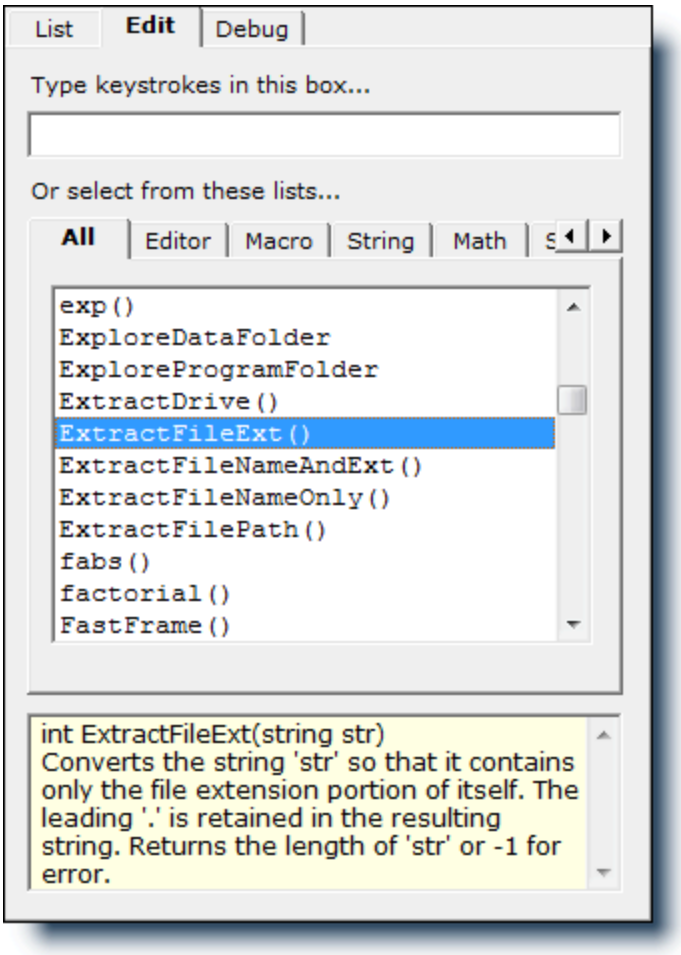
List Tab

New
Use the *New* button to start a new macro. A new macro is created and control will switch to the *Edit Tab*. You will be able to name the macro later when you select the *Save* option.

Copy
The *Copy* button will create a copy of the selected macro. You can then use the *Rename* button to rename the copy, if desired.

Rename
Use the *Rename* button to rename the selected macro.

Delete
Use the *Delete* button to delete the selected macro. A confirmation prompt will be supplied before the macro is deleted.



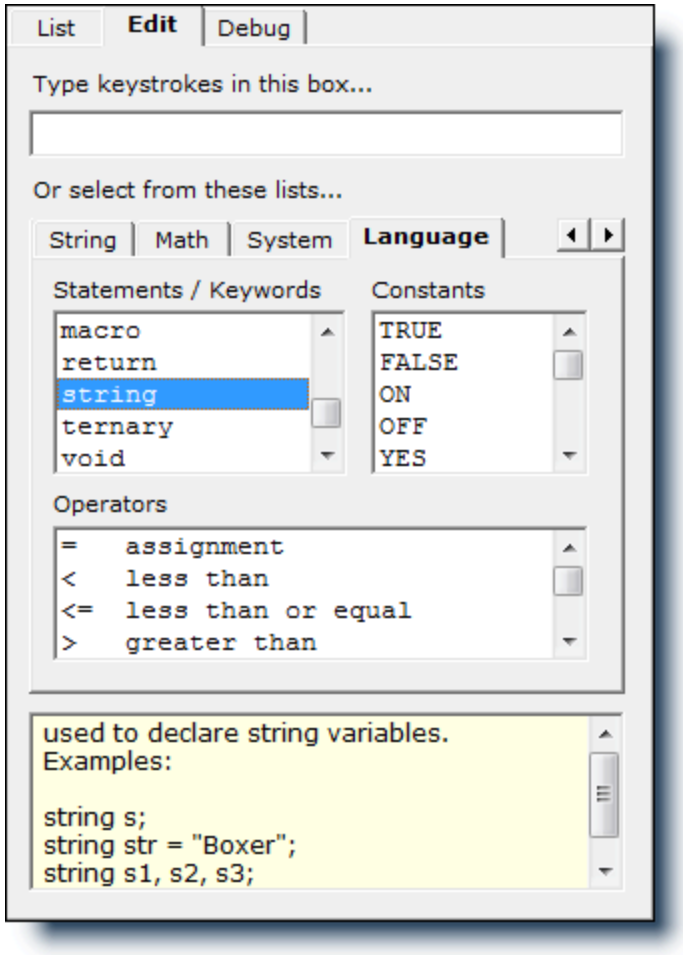
Edit Tab

The Edit Tab contains controls that can be used when composing a macro. If the macro is to be recorded 'by example', simply type the desired keys in the edit box at the top of the panel. You'll notice that the code of the macro is written automatically, as you type, in the editor window at the right. Feel free to switch to the edit window if changes are needed to the macro code. You can resume recording 'by example' at any time by positioning the text cursor in the editor window and returning focus to the edit box at the top left.

When composing a macro by hand, the lists on the *Edit Tab* will prove useful for recalling the macro language function names, keywords, and operators. Each time an entry is selected in a list, the help window at the bottom of the panel displays relevant information about the selected entry. You can insert the selected entry into the editor window by pressing *Enter* or by double-clicking.

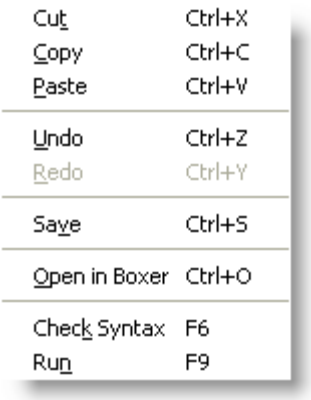
The *All* tab contains a list of all functions that are available in the macro language, regardless of their logical category. The *Editor*, *Macro*, *String* and *Math* tabs display function lists for each of those respective categories. *Editor* contains functions that map to commands available within the editor proper. *Macro* contains functions that are unique to the macro language. *String* contains functions that can be used to manipulate strings. *Math* contains functions that support mathematical operations.

The *Language Tab* contains lists of statements, keywords, constants and operators.

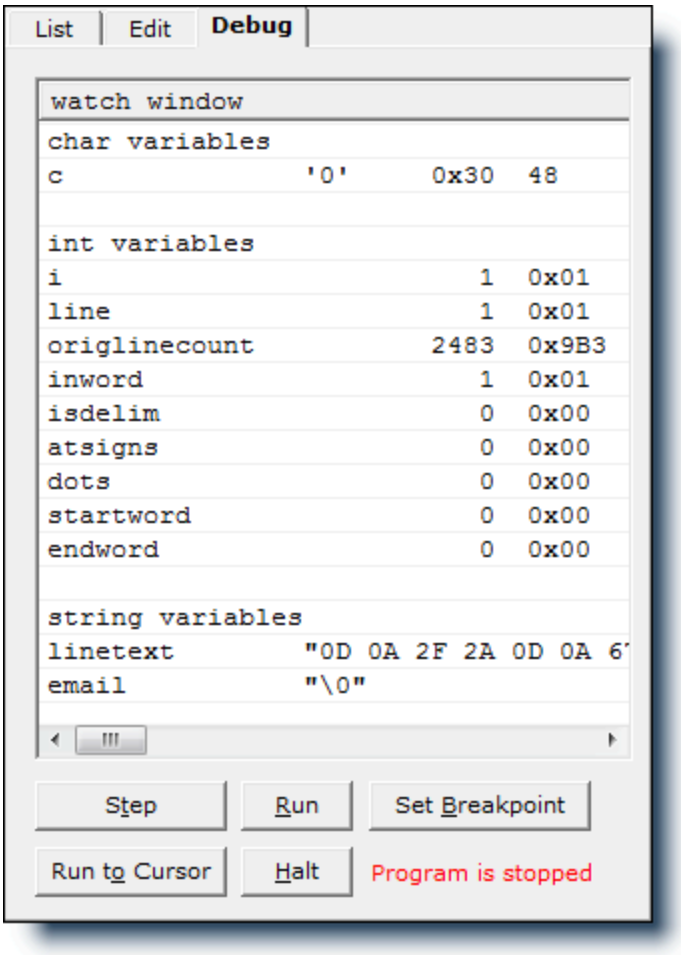


The editor window is used to edit the macro being composed. The macro is displayed with color syntax highlighting, just as if it were being edited in the editor proper. Although the macro editor window looks like a normal Boxer editing window, it is not. You will find that the standard editing and cursor movement commands are available within this window, but Boxer's advanced editing commands are not. If you are composing a complex macro, you might prefer to edit your macro within a normal editing window. For this reason, the editor offers an *Open in Boxer* command on its context menu.

The macro editor window has built-in help for the macro language. You can press *F1* when the cursor is sitting on a function name to view pop-up information for that function.



Additional editor functionality is available on the context menu by right clicking in the editor window when the *Edit Tab* is active.

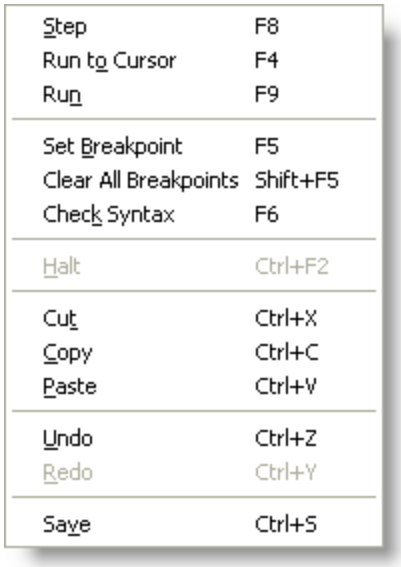


Debug Tab

The *Debug Tab* contains Boxer's integrated macro debugger. The debugger can be used

to control the execution of a macro and view a macro's variables as the macro is executed. The *Watch Window* shows a macro's variables, arranged by type, in both decimal and hexadecimal format. As the macro is executed, the *Watch Window* updates to show the current value of each variable. Note that there is no need to designate a variable as a watch variable; all variables are automatically added to the *Watch Window* each time the macro is debugged.

To begin debugging a macro, click the *Step* button. The *Step* button is used to execute a single line of code. You can click *Step* repeatedly to walk through the macro, line by line. To jump ahead in the macro, position the cursor in the editor window on the line of interest and click the *Run to Cursor* button. Execution will continue until the desired line is reached. The *Run to Cursor* button can be thought of as a one-time breakpoint. To ensure that execution will stop on a selected line every time, use the *Set Breakpoint* button. The line of interest will be highlighted in the editor window with a 'B'. To run the macro without single-stepping, click the *Run* button. *Run* causes the macro to run without interruption, until a breakpoint is hit. If no breakpoints are encountered, the macro will run to completion. Use the *Halt* button to terminate the execution of a macro.



Additional debugger functionality is available on the context menu by right clicking in the editor window when the *Debug* tab is active (see context menu above).

Assigning a Macro to a Key Sequence

There is theoretically no limit to the number of macros that can be created. All of the macros in Boxer's 'Macros' directory will be displayed in the macro list that appears on the *List* tab, and these macros can be run by clicking the nearby *Run* button. In addition, up to 50 macros will be displayed on the Tools | Run Macro submenu, and these macros can be executed directly from that menu. When more than 50 macros are present, those which sort lowest alphabetically will be the first to be omitted from the Run Macro submenu. If you want to force a certain macro to appear in the menu, you can do so by changing its filename to one that will rank higher in an alphabetic

sort.

You may wish to assign commonly used macros to a key assignment to make them easier to execute. There are 50 editor commands available for this use, named *Run Macro 1* to *Run Macro 50*. These commands appear in the command list on the Configure | Keyboard dialog. **In order to make a macro eligible for key assignment, its filename must end with a value from 1 to 50**. For example, if you name a macro `ProcessPayroll24.bm`, that macro can later be run by the key sequence that has been assigned to the *Run Macro 24* command.

Initially, the 50 *Run Macro N* commands are unassigned. Assigning a macro to a given key sequence is thus a two-step process:

1. Make sure that the filename of the macro ends with a value in the range 1-50, and does not conflict with other numbered macros.

2. Use the Configure | Keyboard dialog to assign a key sequence to the corresponding *Run Macro N* command.


If you rename a macro filename from `MyMacro4.bm` to `MyMacro12.bm`, the associated key assignment does not move automatically. The key assignment for the *Run Macro 4* command will always run whatever macro is numbered as 4. Therefore, you will need to visit the Configure | Keyboard dialog to make an adjustment after changing a macro's number.


A macro **cannot** be run from its assigned key sequence if that macro does not appear in the Tools | Run Macro submenu.

Running a Macro from the Command Line

A macro file can be run by naming it on the command line using the -M command line option flag. Please see the notes in that section for full details on this capability.

Running a Macro Automatically on Startup

There may be times when you want Boxer to perform a series of commands--or react to one or more configuration changes--every time the editor is launched. If a macro of the name `startup.bm` is found in the macros directory, it will be run automatically on startup.

Running a Series of Macros in Batch Mode

For some editing tasks it may be desirable to develop a series of macros to perform the necessary conversions. This approach may be desirable when the overall conversion is too complex to implement in a single macro, or when some steps of the conversion will need to be applied selectively on a case-by-case basis. If you have developed a set of macros, say `step1.bm`, `step2.bm` and `step3.bm`, these macros can be run in series from a *macro batch file* -- which might be named `do_it_all.bm --` and which names these files in succession:

```
step1.bm
```

```
step2.bm
step3.bm
```

Blank lines may appear within a *macro batch file*, but all other lines must contain the name of an existing macro file which is to be run.


For a clever tip that tells how to make use of your old Boxer/DOS, Boxer/TKO or Boxer/OS2 macros from within the Windows version of Boxer, see the tip near the bottom of the User Tools topic.

Storing Macro Variables From Run to Run

After a macro has completed, its variables are no longer available for study or use. Two macro functions can be used to store and recall macro variables so that they can be used again at a later time:

```
int WriteValue(string name, char/int/string/float val)
  Writes 'val' to the macro variable storage area named 'name'.
  'name' will be visible to other macros, so be careful to choose
  a unique identifier. Returns 1 for success or -1 for error.
  See also ReadValue(), EraseValue().

int ReadValue(string name, char/int/string/float val)
  Reads a value from the macro variable storage area named 'name'
  and places it into variable 'val'. The type of 'val' must agree
  with the type used when the value was written using WriteValue().
  Returns 1 for success or -1 for error.
  See also WriteValue(), EraseValue().
```

5.156 Macro Examples

The following example macros show the syntax of Boxer's macro language, while also suggesting useful methods of attack for common programming tasks:

Move cursor to bottom of paragraph

```
// move the cursor to the bottom line of the current paragraph

macro BottomOfParagraph()
{
while (LineNumber < LineCount && !LineIsEmpty(LineNumber+1))
   Down;
}
```

Move cursor to top of previous paragraph

```
// move the cursor to the top line of the previous paragraph

macro TopOfPreviousParagraph()
{
Up;

while (LineNumber > 1 && !LineIsEmpty(LineNumber-1))
   Up;

StartOfLine;
}
```

Move cursor to top of current paragraph

```
// move the cursor to the top line of the current paragraph

macro TopOfCurrentParagraph()
{
while (LineNumber > 1 && !LineIsEmpty(LineNumber-1))
   Up;
}
```

Move cursor to top of next paragraph

```
// move the cursor to the first line of the next paragraph

macro TopOfNextParagraph()
{
while (LineNumber < LineCount && !LineIsEmpty(LineNumber))
   Down;

Down;
StartOfLine;
}
```

Add a newline after every closing angle bracket

```
// add a newline after each closing angle (>) character
// unless the angle already appears at end of line
```

```
macro AddNewlineAfterCloseAngle()
{
int line, i, j;
string str;

// loop on all lines in the file
for (line = 1; line <= LineCount(); line++)
   {
   // get the text of line 'line' into string 'str'
   GetLineText(line, str);

   // get the index of the closing angle
   j = strchr(str, '>');

   // if the character was found and was not at end-of-line...
   if (j != -1 && str[j+1] != '\0')
      {
      GotoLine(line);
      GotoColumn(1);

     // advance the cursor to the character
      while (ValueAtCursor() != '>')
         Right;

      // and past the character
      Right;

      // insert a newline
      Enter;

      // process this line again in case other tags exist
      line--;
      }
   }
}
```

Apply HTML markup to a simple text file

```
// apply HTML markup to a simple text file
// also converts double quote, ampersand, and
// angle brackets to HTML equivalents

macro ApplyHTMLMarkup()
{
```

```
int prevlen, len, i;
string str;
int numchanges;

// loop on all lines in the file
for (i = 1; i <= LineCount; i++)
   {
   // get the text of line 'i' into 'str'
   GetLineText(i, str);

   // reset the change counter
   numchanges = 0;

   // convert sensitive characters to HTML codes
   numchanges += ChangeString(str, "&",  "&amp;");
   numchanges += ChangeString(str, "<",  "&lt;");
   numchanges += ChangeString(str, ">",  "&gt;");
   numchanges += ChangeString(str, "\"", "&quot;");

   // if changes were made, replace the line's text
   if (numchanges > 0)
      PutLineText(i, str);
   }

// move to top of file
StartOfFile;
Down;

// loop on all lines in the file, starting on line 2
for (i = 2; i <= LineCount; i++)
   {
   // get the length of the previous line
   prevlen = LineLength(i-1);

   // get the length of this line
   len = LineLength(i);

   // if this line is empty, and the previous line isn't...
   // apply <br> markers to the end of the line
   if (len == 0 && prevlen != 0)
      {
      Up;
      EndOfLine;
      PutString("<br><br>");
      StartofLine;
      Down;
```

```
    }

  Down;         // move down to the next line
  }

StartOfFile;

PutString("<html>\n");
PutString("<head>\n");
PutString("<title></title>\n");
PutString("</head>\n\n");
PutString("<body>\n");

EndOfFile;
EndOfLine;
PutString("\n");
PutString("</body>\n");
PutString("</html>\n");

// place cursor between title and /title
GotoLine(3);
GotoColumn(8);
}
```

Display an ASCII chart in a new file

```
// ASCII chart example

macro ASCIIchart(void)
{
char i;

// open a new file
New;

// loop from space to 255 to show all chars
for (i = ' '; i <= 255; i++)
  printf("The ASCII value of '%c' is %d\n", i, i);
}
```

Convert comma-separated-value (CSV) data

```
// convert comma-separated-value (CSV) data on the current
```

```
// line so that each field is placed on its own line

macro ConvertCSV()
{
string str;
int numquotes, numcommas;

// get the count of quotes/commas on this line
numquotes = LineContains(linenumber, "\"");

numcommas = LineContains(linenumber, ",");

// if this appears to be CSV data...
if (numcommas+1 == numquotes / 2)
   {
   // get the text of the current line
   GetLineText(linenumber, str);

   // remove any empty data fields
   ChangeString(str, "\"\",", "");

   // convert "," to a newline
   ChangeString(str, "\",\"", "\n");

   // remove the first and last quotes
   ChangeString(str, "\"", "");

   // select the line
   GoToColumn(1);
   SelectToEndOfLine;

   // replace the selection
   PutString(str);
   }

// position for next line
Down;
StartOfLine;
}
```

Cut lines containing a user-defined string

```
// cut lines containing a user-defined string to the Windows clipboard
```

```
macro CutLinesContaining();
{
int line;
int len;
string str;
int numcut = 0;

// get the string from the user
len = GetString("Cut lines containing this string:", str);

if (len == 0)
   return;

// make the Windows clipboard the active clipboard
SetClipboard(0);

// clear the Windows clipboard
ClearClipboard(0);

// move cursor to start of file
StartOfFile();

// loop on all lines in the file
for (line = 1; line <= LineCount(); line++)
   {
   // does this line contain the string?
   if (LineContains(line, str))
      {
      GotoLine(line);
      CutAppend();
      numcut++;          // tally the cut
      line--;            // stay here for next line
      }
   }

// report the results
if (numcut == 1)
   message("Results", "1 line was cut to the Windows clipboard");
else
   message("Results", numcut,
           " lines were cut to the Windows clipboard");
}
```

Delete blank lines

```
// delete blank lines in the current file

macro DeleteBlankLines(void)
{
int i, len;

// start at the top of the file
StartOfFile;

// loop on all lines in the file
for (i = 1; i <= LineCount; i++)
   {
   // get the length of this line
   len = LineLength(i);

   // is this line empty?
   if (len == 0)
      {
      DeleteLine;     // delete this line
      i--;            // stay at this line #
      }
   else
      {
      Down;           // move down to the next line
      }
   }
}
```

Delete lines containing a user-defined string

```
// deletes lines containing a user-defined string

macro DeleteLinesContaining()
{
int line;
int len;
string str;
int deleted = 0;

// get the string from the user
len = GetString("Delete lines containing this string:", str);
```

```
if (len == 0)
   return;

// loop on all lines in the file
for (line = 1; line <= LineCount(); line++)
   {
   // does this line contain the string?
   if (LineContains(line, str))
      {
      DeleteLine(line);   // delete it
      deleted++;          // tally the deletion
      line--;             // stay here for next line
      }
   }

// report the results
if (deleted == 1)
   message("Results", "1 line was deleted");
else
   message("Results", deleted, " lines were deleted");
}
```

Delete lines NOT containing a user-defined string

```
// deletes lines NOT containing a user-defined string

macro DeleteLinesNotContaining()
{
int line;
int len;
string str;
int deleted = 0;

// get the string from the user
len = GetString("Delete lines that do NOT contain this string:", str);

if (len == 0)
   return;

// loop on all lines in the file
for (line = 1; line <= LineCount(); line++)
   {
   // does this line contain the string?
   if (!LineContains(line, str))
```

```
        {
        DeleteLine(line);   // delete it
        deleted++;          // tally the deletion
        line--;             // stay here for next line
        }
    }

// report the results
if (deleted == 1)
    message("Results", "1 line was deleted");
else
    message("Results", deleted, " lines were deleted");
}
```

Compute return on a deposited amount

```
// Compute result of amount left on deposit with continuous
// compounding.  Uses the formula: P = pe^rt

macro ComputeDeposit()
{
float amt, newamt, rate, years;
string str;

GetFloat("Enter the amount on deposit:", amt);

GetFloat("Enter the interest rate:", rate);

// if user entered 5, make it .05, for example
if (rate > 1.0)
    rate /= 100.0;

GetFloat("Enter the number of years on deposit:", years);

newamt = amt * pow(e, rate * years);

sprintf(str, "The amount with interest applied is: %.2f", newamt);
Message("Result", str);
}
```

Add blank lines after lines ending with !.?

```
// add a blank line after any line that ends with !.?

macro AddBlankLines(void)
{
char ch;
int i;

// loop on all lines in the file
for (i = 1; i <= LineCount; i++)
   {
   // make sure this line is not empty
   if (LineLength(i) > 1)
      {
      // move the cursor to this line
      GotoLine(i);

      // move to the end of the line
      EndOfLine;

      // backup off newline and onto last char
      Left;

      // get the value of char at the cursor
      ch = ValueAtCursor();

      // if it's a line ender, add Enter
      if (ch == '.' || ch == '?' || ch == '!')
         {
         EndOfLine;
         Enter;
         }
      }
   }
}
```

Double space and reformat

```
// double space and reformat the text on the clipboard
// prepares a web document for printing

macro DoubleSpaceAndReformat(void)
{
int i;
int numlines;
```

```
// save various editor settings
SaveSettings;

// open a new file and paste from clipboard
New;
Paste;

// set Text Width to 96
TextWidth(96);

// delete all blank lines
DeleteBlankLines;

// record the number of lines BEFORE we start adding lines
numlines = LineCount - 1;

// go to the top
StartOfFile;

// double space the file
for (i = 1; i <= numlines; i++)
   {
   Down;
   PutString("\n");
   }

// reformat the whole file
SelectAllText;
Reformat;

// remove any small indents that might be present
SelectAllText;
for (i = 1; i <= 12; i++)
Unindent;

// release the selection
Deselect;

// restore various editor settings
RestoreSettings;
}
```

Extract email addresses

```
// extract email addresses from all lines within the current file
// and append them to the end of the file

macro ExtractEmailAddresses()
{
int i, line, origlinecount;
int inword, isdelim;
int atsigns, dots;
int startword, endword;
string linetext, email;
char c;

// note the linecount before we start adding more lines
origlinecount = LineCount;

// loop on all lines in the file
for (line = 1; line <= origlinecount; line++)
   {
   // get the text of the whole line into the string 'linetext'
   GetLineText(line, linetext);

   // add a space to make end-of-line handling smoother
   strcat(linetext, " ");

   // loop on all characters in 'linetext'
   for (inword = FALSE, i = 0; linetext[i] != 0; i++)
      {
      c = linetext[i];

      // set a flag if this character is one that delimits words
      if (isalnum(c) || (strchr("_@.-", c) != -1))
         isdelim = FALSE;
      else
         isdelim = TRUE;

      // decide whether this character starts a new word,
      // or ends an existing word
      if (inword && isdelim)
         {
         inword = FALSE;
         endword = i-1;

         // we've just left a word: see if it had both
         // the required characters
         if (atsigns == 1 && dots >= 1)
            {
```

```
            // get the linetext address into a string
            SubString(email, linetext, startword, endword-startword+1);

            // add it to the end of the file
            EndOfFile;
            EndOfLine;
            Enter;
            PutString(email);
            }
         }
      else if (!inword && !isdelim)
         {
         inword = TRUE;
         startword = i;
         atsigns = 0;
         dots = 0;
         }

      // tally whether or not we see the required chars while we're in a
      if (inword)
         {
         if (linetext[i] == '@')
            atsigns++;

         if (linetext[i] == '.')
               dots++;
         }
      }
   }
}
```

Hex to Decimal

```
// shows hex to decimal conversion technique

macro HexToDecimal()
{
string str;
int x = 0;
int i, val;
char ch;

GetString("Enter a hexadecimal string", str);
```

```
for (i = 0; str[i] != '\0'; i++)
   {
   ch = str[i];

   if (!isxdigit(ch))
      {
      message("Error", "Invalid character encountered: ", ch);
      return;
      }

   ch = toupper(ch);

   if (isalpha(ch))
      val = ch - 'A' + 10;
   else
      val = ch - '0';

   x = x * 16;
   x = x + val;
   }

message("Result", "The decimal value is ", x);
}
```

Obfuscate the selected text with HTML codes

```
// convert the word selected into its HTML coded format

// this can be used to convert phone numbers and email addresses
// in web pages to frustrate automated crawlers from harvesting
// your information for spam lists

macro Obfuscate()
{
int i;
string str;
string result;
string tmp;

if (!TextIsSelected)
   {
   Message("Error",
      "Please select a word before\nrunning the macro.\n");
   return;
```

```
    }

GetSelection(str);

// loop on all characters in 'str'
for (i = 0; str[i] != '\0'; i++)
    {
    sprintf(tmp, "&#%03d;", str[i]);
    strcat(result, tmp);
    }

PutSelection(result);
}
```

Display 24-hour time

```
// display the current time in 24-hour time format

macro Print24HourTime()
{
int h, m, s;

GetTime24(h, m, s);
printf("%d:%02d:%02d", h, m, s);
}
```

Reduce blank lines

```
// reduce multiple blank lines to one blank line

macro ReduceBlankLines(void)
{
int thislen, prevlen, i;

// position cursor to line 2
StartOfFile;
Down;

// loop on all lines in the file (starting with line 2)
for (i = 2; i <= LineCount; i++)
    {
    // get the length of the previous line
    prevlen = LineLength(i-1);
```

```
    // get the length of this line
    thislen = LineLength(i);

    // are both previous and this line empty?
    if (prevlen == 0 && thislen == 0)
        {
        DeleteLine;     // delete this line
        i--;            // stay at this line #
        }
    else
        {
        Down;           // move down to the next line
        }
    }
}
```

Reformat to an alternative text width

```
// Reformat the current paragraph to 70 characters, regardless
// of what the current Text Width setting is

macro ReformatAlternative()
{
SaveSettings;
TextWidth(70);
Reformat;
RestoreSettings;
}
```

Extract double quoted strings

```
// extract double quoted strings from the current file
// and append them at the bottom of the file

macro ExtractStrings()
{
string s, s1, s2, s3;
int i, j, k;
int found = 0;
int original_linecount = LineCount();
```

```
// loop on all lines in the current file
for (i = 1; i <= original_linecount; i++)
   {
   // does this line have two or more double quotes?
   while (LineContains(i, "\"") >= 2)
      {
      // tally number of strings found
      found++;

      // get the text of line 'i' into string 's'
     GetLineText(i, s);

      // get the offset of the first double quote
      j = strchr(s, '\"');

      // get the index of the second double quote
      for (k = j + 1; s[k] != '\"'; k++)
         ;

      // get the first portion into 's1'
      SubString(s1, s, 0, j);

      // get the second portion (the string) into 's2'
     SubString(s2, s, j, k-j+1);

      // get the third portion into 's3'
     SubString(s3, s, k+1, 2048);

      // build the new line and replace it
      s = s1;
      s += s3;
     PutLineText(i, s);

      // gather the strings at the bottom of the current file
     EndOfFile();
     EndOfLine();
     printf("\n%s", s2);
      }
   }

// report the results
if (found == 1)
   printf("\n\n%d string was found and removed\n", found);
else
   printf("\n\n%d strings were found and removed\n", found);
}
```

Reverse the text of each line

```
// reverse the text on every line in the file
// "abcdefg" becomes "gfedcba"

macro ReverseLineText()
{
int i, len, line;
string str;
char tmp;

// loop on all lines in the file
for (line = 1; line <= LineCount; line++)
   {
   len = linelength(line);

   // ignore lines too short/long
   if (len >= 2 && len < 2000)
      {
      // get the text of line 'line' into string 'str'
     GetLineText(line, str);

      // loop through half this line
     for (i = 0; i < len/2; i++)
         {
         // swap the characters...
         tmp = str[i];
         str[i] = str[len-1-i];
         str[len-1-i] = tmp;
         }

      // replace line with reversed line
      PutLineText(line, str);
      }
   }
}
```

Reverse names: Smith.Bob to Bob.Smith

```
// changes a list of "Smith.Bob" entries to "Bob.Smith"
```

```
macro ReverseNames()
{
int line, i;
string str;
string first, last;
string newstring;

// loop on all lines in the file
for (line = 1; line <= LineCount; line++)
   {
   // get the text of line 'line' into string 'str'
   GetLineText(line, str);

   // look for a '.' within 'str'
   i = strstr(str, ".");

   if (i != -1)
      {
      // 'last' gets 'i' chars from 'str' starting at index 0
      SubString(last,  str, 0,   i);

      // 'first' gets up to 100 chars from 'str' starting at index i+1
      SubString(first, str, i+1, 100);

      // build a new string from 'first' and 'last'
      sprintf(newstring, "%s.%s", first, last);

      // replace the text of the line
      PutLineText(line, newstring);
      }
   }
}
```

Truncate lines after a user-defined string

```
// truncate lines after a user-defined string

macro TruncateLineAfterString()
{
int j, line, len;
int truncated = 0;
string str, linestr, newstr;

// get the string from the user
```

```
len = GetString("Truncate lines after this string:", str);

// if the string is empty, quit
if (len == 0)
        return;

// loop on all lines in the file
for (line = 1; line <= LineCount; line++)
   {
   GetLineText(line, linestr);

   // does this line contain the string?
   if ((j = strstr(linestr, str)) != -1)
      {
      // create a new string without the trailing text
      SubString(newstr, linestr, 0, j+len);

      // replace this line with the new text
      PutLineText(line, newstr);

      // tally the truncation
      truncated++;
      }
   }

// report the results
if (truncated == 1)
   message("Results", "1 line was truncated");
else
   message("Results", truncated, " lines were truncated");
}
```

Truncate lines at a user-defined string

```
// truncate lines at a user-defined string

macro TruncateLineAtString()
{
int j, line, len;
int truncated = 0;
string str, linestr, newstr;

// get the string from the user
len = GetString("Truncate lines at this string:", str);
```

```
// if the string is empty, quit
if (len == 0)
        return;

// loop on all lines in the file
for (line = 1; line <= LineCount; line++)
   {
   GetLineText(line, linestr);

   // does this line contain the string?
   if ((j = strstr(linestr, str)) != -1)
      {
      // create a new string without the trailing text
      SubString(newstr, linestr, 0, j);

      // replace this line with the new text
      PutLineText(line, newstr);

      // tally the truncation
      truncated++;
      }
   }

// report the results
if (truncated == 1)
   message("Results", "1 line was truncated");
else
   message("Results", truncated, " lines were truncated");
}
```

Delete lines that begin with a user-defined string

```
// deletes lines that begin with a user-defined string

macro DeleteLinesThatBeginWith()
{
int line, len;
string str, linestr;
int deleted = 0;

// get the string from the user
len = GetString("Delete lines that begin with:", str);
```

```
if (len == 0)
   return;

// loop on all lines in the file
for (line = 1; line <= LineCount(); line++)
   {
         // get the text of line 'line' into 'linestr'
         GetLineText(line, linestr);
            // does this line start with 'str'?
         if (strncmp(linestr, str, len) == 0)
               {
               DeleteLine(line);   // delete it
               deleted++;          // tally the deletion
               line--;             // stay here for next line
               }
         }

// report the results
if (deleted == 1)
   message("Results", "1 line was deleted");
else
   message("Results", deleted, " lines were deleted");
}
```

Delete lines that end with a user-defined string

```
// deletes lines that end with a user-defined string

macro DeleteLinesThatEndWith()
{
int line, len, linelen;
string str, linestr, str2;
int deleted = 0;

// get the string from the user
len = GetString("Delete lines that end with:", str);

if (len == 0)
   return;

// loop on all lines in the file
for (line = 1; line <= LineCount(); line++)
   {
   // get the text of line 'line' into 'linestr'
```

```
    linelen = GetLineText(line, linestr);

    // if the line is too short, do nothing
    if (linelen < len)
       {
       ;
       }
    // does this line end with 'str' ?
    else
       {
       // isolate the tail of the line into a string
       SubString(str2, linestr, linelen - len, len);

       if (strcmp(str2, str) == 0)
          {
          DeleteLine(line);   // delete it
          deleted++;          // tally the deletion
          line--;             // stay here for next line
          }
       }
    }

// report the results
if (deleted == 1)
   message("Results", "1 line was deleted");
else
   message("Results", deleted, " lines were deleted");
}
```

Call MapQuest to show a map

```
// get an address from the user and call it up on MapQuest

macro CallMapQuest()
{
string city, state, address, country, url;
int zoom = 7;

// get information from the user
GetString("Enter street address:", address);
GetString("Enter city/town:",      city);
GetString("Enter state/province:", state);
GetString("Enter country:",        country);
```

```
// state level maps are better at zoom level 3
if (city == "")
   zoom = 3;

// country level maps are better at zoom level 1
if (state == "")
   zoom = 1;

// convert embedded spaces to plus signs
ChangeString(address, " ", "+");
ChangeString(city," ", "+");
ChangeString(state,   " ", "+");
ChangeString(country, " ", "+");

// build the URL that will be used
sprintf(url,
"http://www.mapquest.com/maps/map.adp?city=%s&state=%s&address=%s&c
ountry=%s&zoom=%d"    , city, state, address, country, zoom);

// send the URL to Windows so the default browser is run
OpenURL(url);
}
```

Convert British English punctuation to American English punctuation

```
// Convert British English punctuation to American English punctuation
// (in Britain, periods and commas are placed outside double quotes)

macro BritishPunctuation()
{
// notice that the double quote character must be escaped with a
// backslash when it appears within a string

// change ".  to  ."
ReplaceAll("\".",  ".\"");

// change ",  to  ,"
ReplaceAll("\"\,", ",\"");
}
```

5.157 Macro Function Reference

Function	Prototype and Description
abs()	int abs(int n) Returns the absolute value of 'n'.
acos()	float acos(float x) Returns the arc cosine of 'x' in radians. 'x' must be in the range -1 to 1.
ActiveClipboard	int ActiveClipboard Returns the number of the active clipboard. The Windows Clipboard is number 0; private clipboards are numbered 1 to 8.
ActiveSpellChecking()	int ActiveSpellCheck(int mode) Enables or disables Active Spell Checking according to 'mode'.
AlignCenter	Issues the Align Center command
AlignLeft	Issues the Align Left command
AlignRight	Issues the Align Right command
AlignSmooth	Issues the Align Smooth command
ANSIChart	Issues the ANSI Chart command
ANSItoOEM	Issues the ANSI to OEM command
Append	Append Append the selected text to the current clipboard. If no text is selected, the current line is appended to the clipboard.
AppendToClipboard()	int AppendToClipboard(string str, int n) Appends string 'str' to Clipboard 'n'. Returns the total length of the text on the clipboard, or -1 for error. The Windows Clipboard is number 0; private clipboards are numbered 1 to 8. See also PutClipboardText().
ApplyHighlighting	Issues the Apply Highlighting command
ArrangeIcons	Issues the Arrange Icons command
ASCIItoEBCDIC	Issues the ASCII to EBCDIC command
asin()	float asin(float x) Returns the arc sine of 'x' in radians. 'x' must be in the range -1 to 1.
atan()	float atan(float x) Returns the arc tangent of 'x' in radians.

atof()	float atof(string str) Returns the floating point value of the number described by string 'str'.
atoi()	int atoi(string str) Returns the integer value of the decimal number described by string 'str'.
AutoNumber	Issues the Auto-Number command
Backspace	Issues the Backspace command
Backtab	Issues the Backtab command
Beep()	Beep(int freq, int duration) Makes a sound through the PC speaker using the supplied values for frequency and duration. Frequency is in Hz and duration is in milliseconds. Beep(1000, 300) produces a standard beep.
BookmarkManager	Issues the Bookmark Manager command
BottomOfPage	Issues the Bottom of Page command
BringUserListsToTop	Issues the Bring User Lists to Top command
BrowseForFilename()	BrowseForFilename(string fn, int mustexist) Browse for a filename using a standard Windows open dialog and place the selected filename in 'fn'. If 'mustexist' is non-zero, the selected filename must already exist. If 'mustexist' is 0, a new filename can be selected. Returns 1 for success or -1 for error.
ByteCount	int ByteCount Returns the number of characters in the current file.
Calculator	Issues the Calculator command
Calendar	Issues the Calendar command
Cascade	Issues the Cascade command
CascadeHorizontal	Issues the Cascade Horizontal command
CascadeVertical	Issues the Cascade Vertical command
CaseInvert	Issues the Case Invert command
CaseLower	Issues the Case Lower command
CaseSentences	Issues the Case Sentences command
CaseTitle	Issues the Case Title command
CaseUpper	Issues the Case Upper command
CaseWords	Issues the Case Words command
ceil()	float ceil(float x) Returns (as a float) the smallest integer not less than 'x'. Example: ceil(1.5) returns 2.0.

ChangeString()	int ChangeString(string str1, str2, str3) Searches 'str1' and changes all occurrences of 'str2' to the string 'str3'. Returns the number of changes made or -1 for error. The search is case sensitive. Regular expressions are not recognized. If 'str3' is an empty string, the effect will be to delete all occurrences of 'str2' within 'str1'.
ChangeStringi()	int ChangeStringi(string str1, str2, str3) Searches 'str1' and changes all occurrences of 'str2' to the string 'str3'. Returns the number of changes made or -1 for error. The search is case insensitive. Regular expressions are not recognized. If 'str3' is an empty string, the effect will be to delete all occurrences of 'str2' within 'str1'.
ChangeStringRE()	int ChangeStringRE(string str1, str2, str3) Searches 'str1' and changes all instances matching 'str2' to the string 'str3'. Returns the number of changes made or -1 for error. The search is case sensitive. Regular expressions ARE recognized in 'str2'. If 'str3' is an empty string, the effect will be to delete all occurrences of 'str2' within 'str1'.
ChangeStringREi()	int ChangeStringREi(string str1, str2, str3) Searches 'str1' and changes all instances matching 'str2' to the string 'str3'. Returns the number of changes made or -1 for error. The search is case insensitive. Regular expressions ARE recognized in 'str2'. If 'str3' is an empty string, the effect will be to delete all occurrences of 'str2' within 'str1'.
CheckWord	Issues the Check Word command
ClearAllBookmarks	Issues the Clear All Bookmarks command
ClearAllClipboards	Issues the Clear All Clipboards command
ClearClipboard()	ClearClipboard(int n) Clears the content of Clipboard 'n'. The Windows Clipboard is number 0; private clipboards are numbered 1 to 8.
ClearClosedTabsList	Issues the Clear Closed Tabs List
ClearRecentFilesList	Issues the Clear Recent Files List command
ClearRecentProjectsList	Issues the Clear Recent Projects List command
ClearUndo	Issues the Clear Undo command
Close	Issues the Close command
CloseAll	Issues the Close All command
CloseAllButActive	Issues the Close All But Active command
ColorChart	Issues the HTML Color Chart command

Column	int Column Returns the column number of the text cursor in the current file, or -1 for error. The column returned is 1-based, not 0-based, and does not give consideration to the display value of any tabs that may appear in the line. See also DisplayColumn().
Comment	Issues the Comment command
ConfigureColors	Issues the Configure Colors command
ConfigureCtagsFunctionIndexing	Issues the Configure Ctags Function Indexing command
ConfigureKeyboard	Issues the Configure Keyboard command
ConfigurePreferences	Issues the Configure Preferences command
ConfigurePrinterFont	Issues the Configure Printer Font command
ConfigureScreenFont	Issues the Configure Screen Font command
ConfigureSyntaxHighlighting	Issues the Configure Syntax Highlighting command
ConfigureTemplates	Issues the Configure Templates command
ConfigureTextHighlighting	Issues the Configure Text Highlighting command
ConfigureToolbar	Issues the Configure Toolbar command
ConfigureUserTools	Issues the Configure User Tools command
Copy	Copy Copy the selected text to the current clipboard. If text is not selected, the current line is copied to the clipboard.
CopyFile()	int CopyFile(string oldname, string newname) Copies the file 'oldname' to the file 'newname', overwriting the output file if it already exists. Returns 1 for success or -1 for error.
CopyFilename	Issues the Copy Filename command
cos()	float cos(float x) Returns the cosine of 'x'. The angle 'x' must be in radians.
cosh()	float cosh(float x) Returns the hyperbolic cosine of 'x'. The angle 'x' must be in radians.
CreateDirectory()	int CreateDirectory(string dir) Creates a new directory according to the fully qualified filepath in 'dir'. Returns 1 for success or -1 for error.

CtagsFunctionIndex	Issues the Ctags Function Index command
Cut	Cut Cut the selected text to the current clipboard. If text is not selected, the current line is cut to the clipboard.
CutAppend	CutAppend Cut the selected text and append it to the current clipboard. If text is not selected, the current line is cut and appended to the clipboard.
Declaration	Issues the Declaration command
Decrement()	int Decrement(int n) Subtracts 'n' from the value at the text cursor and places the result in the text file. Returns the result of the operation or -1 for error. If 'n' is not supplied the Decrement dialog will appear when the macro is run.
Delete	Delete Deletes the character at the text cursor, or the selected text.
DeleteBlankLines	Issues the Delete Blank Lines command
DeleteBookmarkedLines	Issues the Delete Bookmarked Lines command
DeleteDuplicateLines	Issues the Delete Duplicate Lines command
DeleteFile()	int DeleteFile(string name) Deletes the fully qualified filepath 'name' from the disk, without requesting confirmation. Returns 1 for success or -1 for error.
DeleteLine	int DeleteLine(int n) Deletes line 'n' in the current file. Returns 1 for success or -1 for error. If 'n' is not supplied, the current line is deleted.
DeleteLinesThatBeginWith	int DeleteLinesThatBeginWith(string str) Delete lines that begin with the string 'str'. Returns 1 for success or -1 for error. If 'str' is not supplied a dialog will appear when the macro is run.
DeleteLinesThatContain	int DeleteLinesThatContain(string str) Delete lines that contain the string 'str'. Returns 1 for success or -1 for error. If 'str' is not supplied a dialog will appear when the macro is run.
DeleteLinesThatDoNotBeginWith	int DeleteLinesThatDoNotBeginWith(string str) Delete lines that do not begin with the string 'str'. Returns 1 for success or -1 for error. If 'str' is not supplied a dialog will appear when the macro is run.
DeleteLinesThatDoNotContain	int DeleteLinesThatDoNotContain(string str) Delete lines that do not contain the string 'str'.

	Returns 1 for success or -1 for error. If 'str' is not supplied a dialog will appear when the macro is run.
DeleteLinesThatDoNotEndWith	int DeleteLinesThatDoNotEndWith(string str) Delete lines that do not end with the string 'str'. Returns 1 for success or -1 for error. If 'str' is not supplied a dialog will appear when the macro is run.
DeleteLinesThatEndWith	int DeleteLinesThatEndWith(string str) Delete lines that end with the string 'str'. Returns 1 for success or -1 for error. If 'str' is not supplied a dialog will appear when the macro is run.
DeleteNextWord	Issues the Delete Next Word command
DeletePreviousWord	Issues the Delete Previous Word command
DeleteToEndOfLine	Issues the Delete to End of Line command
DeleteToStartOfLine	Issues the Delete to Start of Line command
Deselect()	Deselect(int mode) Releases the text selection, if one exists. If 'mode' is 0, the text cursor is placed at the beginning of the selection. If 'mode' is 1, the text cursor is placed at the end of the selection. If 'mode' is 2, the current position of the text cursor is maintained. Returns 1 for success or -1 for error.
DisplayColumn	int DisplayColumn Returns the column number of the text cursor in the current file, or -1 for error. The column returned is 1-based, not 0-based, and gives consideration to the display value of any tabs that may appear in the line. See also Column().
Divide()	int Multiply(int n) Divides the value at the text cursor by 'n' and places the result in the text file. Returns the result of the operation or -1 for error. If 'n' is not supplied the Divide dialog will appear when the macro is run.
Down	int Down(int n) Issues the Down command 'n' times. Returns the number of commands performed or -1 for error. The argument 'n' is optional; if it is not provided a single command is performed.
DuplicateAndIncrement	Issues the Duplicate and Increment command
DuplicateLine	Issues the Duplicate Line command
e	float e Returns the value of Euler's number 'e', which is approximately 2.7182818285.

EBCDICtoASCII	Issues the EBCDIC to ASCII command
EditClipboard()	EditClipboard(int n) Opens Clipboard 'n' in a window for editing. The Windows Clipboard is number 0; private clipboards are numbered 1 to 8.
EndOfFile	Issues the End of File command
EndOfLine	Issues the End of Line command
Enter	Issues the Enter command
EraseValue()	int EraseValue(string name) Erases the variable 'name' from the macro variable storage area. Returns 1 for success or -1 for error. See also ReadValue(), WriteValue(), ValueExists().
ErrorChart	Issues the Error Chart command
Exit	Exit Issues the File|Exit command to close the editor. If one or more files have not been saved a prompt will appear when the macro is run.
exp()	float exp(float x) Returns the value 'e' raised to the 'x'.
ExploreDataFolder	Issues the Explore Data Folder command
ExploreProgramFolder	Issues the Explore Program Folder command
ExtractDrive()	int ExtractDrive(string str) Converts the string 'str' so that it contains only the drive designation portion of itself (eg 'C:'). Returns the length of 'str' or -1 for error.
ExtractFileExt()	int ExtractFileExt(string str) Converts the string 'str' so that it contains only the file extension portion of itself. The leading '.' is retained in the resulting string. Returns the length of 'str' or -1 for error.
ExtractFileNameAndExt()	int ExtractFileNameAndExt(string str) Converts the string 'str' so that it contains the filename.ext portion of itself. Returns the length of 'str' or -1 for error.
ExtractFileNameOnly()	int ExtractFileNameOnly(string str) Converts the string 'str' so that it contains only the filename portion of itself. Returns the length of 'str' or -1 for error.
ExtractFilePath()	int ExtractFilePath(string str) Converts the string 'str' so that it contains only the filepath portion of itself. The trailing backslash is retained in the resulting string. Returns the length

	of 'str' or -1 for error.
fabs()	float fabs(float x) Returns the absolute value of 'x'.
factorial()	int factorial(int x) Returns the value of x factorial, also known as x! Returns -1 for error or overflow.
FastFrame()	int FastFrame(int style) Surrounds the columnar selection with a frame according to 'style'. When 'style' is in the range 1 to 11, a corresponding line style from the Fast Frame dialog is used. If 'style' is not supplied the Fast Frame dialog will appear when the macro runs. Returns 1 for success or -1 for error.
FileCount	int FileCount Returns the number of files currently open in the editor.
FileExists()	int FileExists(string filepath) Returns 1 if 'filepath' exists, 0 if it does not exists, or -1 for error.
FileName	int FileName(string fn) Fills 'fn' with the full path of the current file. Returns the length of the filepath or -1 for error.
FilePicker	Issues the File Picker command
FileProperties	Issues the File Properties command
FileTabsBottom	Issues the File Tabs Bottom command
FileTabsTop	Issues the File Tabs Top command
FillWithString()	int FillWithString(string str) Fills the selected region with 'str'. Returns 1 for success or -1 for error. If 'str' is not supplied the Fill with String dialog will appear when the macro is run.
Find()	int Find(string str) Searches for string 'str'. Returns TRUE if found, FALSE if not found, -1 for error. If 'str' is not supplied the Find dialog will appear when the macro is run. Find() will use the current settings on the Find dialog, but it will force the Perl Regular Expressions option OFF, and the Match Case option ON. See also Findi(), FindRE() and FindREi().
FindADiskFile	Issues the Find a Disk File command
FindAndCount()	int FindAndCount(string str) Searches for occurrences of 'str' and returns the

	number found. If 'str' is not supplied the Find and Count dialog will appear when the macro is run.
FindDifferingLines	int FindDifferingLines() Issues the Find Differing Lines command and returns 1 if a mismatch is found, or 0 if no additional mismatches exist.
FindDistinctLines	Issues the Find Distinct Lines command
FindDuplicateLines	Issues the Find Duplicate Lines command
FindFast	Issues the Find Fast command
Findi()	int Findi(string str) Searches for string 'str'. Returns TRUE if found, FALSE if not found, -1 for error. If 'str' is not supplied the Find dialog will appear when the macro is run. Findi() will use the current settings on the Find dialog, but it will force the Perl Regular Expressions option OFF, and the Match Case option OFF. See also Find(), FindRE() and FindREi().
FindMate	int FindMate search for a mate to the parenthetical sequence at the text cursor. Returns TRUE if found, FALSE if not found.
FindNext	int FindNext Returns 1 if the string is found, 0 if not found, -1 for error.
FindPrevious	int FindPrevious Returns 1 if the string is found, 0 if not found, -1 for error.
FindRE()	int FindRE(string str) Searches for string 'str'. Returns TRUE if found, FALSE if not found, -1 for error. If 'str' is not supplied the Find dialog will appear when the macro is run. FindRE() will use the current settings on the Find dialog, but it will force the Perl Regular Expressions option ON, and the Match Case option ON. See also Find(), Findi() and FindREi().
FindREi()	int FindREi(string str) Searches for string 'str'. Returns TRUE if found, FALSE if not found, -1 for error. If 'str' is not supplied the Find dialog will appear when the macro is run. FindREi() will use the current settings on the Find dialog, but it will force the Perl Regular Expressions option ON, and the Match Case option

	OFF. See also Find(), Findi() and FindRE().
FindTextInDiskFiles	Issues the Find Text in Disk Files command
FindUniqueLines	Issues the Find Unique Lines command
FlipCase	Issues the Flip Case command
floor()	float floor(float x) Returns (as a float) the largest integer not greater than 'x'. Example: floor(1.5) returns 1.0.
FormatXML	Issues the Format XML command
Formfeed	Issues the Formfeed command
FTPOpen()	int FTPOpen(string fn) Opens the FTP file 'fn' for editing. If 'fn' is already open for editing, its window will become the current window. Returns 1 for success or -1 for error. Remember: \\ must be used to denote \ within 'fn'.
GetChar()	int GetChar(string prompt, char c) Displays a message box with 'prompt' and fills 'c' with the character entered by the user. Returns 1 for success or -1 for error. See also PressChar().
GetClipboardText()	int GetClipboardText(string str, int n) Fills 'str' with the text of Clipboard 'n'. Returns the length of the text installed or -1 for error. The Windows Clipboard is number 0; private clipboards are numbered 1 to 8.
GetCurrentDirectory()	int GetCurrentDirectory(string str) Retrieves the current directory for the active process and places it in 'str'. Returns 1 for success or -1 for error. See also SetCurrentDirectory().
GetDataDirectory()	int GetDataDirectory(string str) Fills 'str' with the full path of the data directory. Returns 1 for success or -1 for error. See also GetProgramDirectory().
GetDate()	int GetDate(int y, int m, int d) Gets the current date and fills 'y', 'm' and 'd' with the year, month and date, respectively. Returns 1 for success or -1 for error.
GetDayName()	int GetDayName(string str, int n) Fills 'str' with the 3-character name of weekday

	number 'n' (1-7). The string returned is sensitive to the local language. Returns 1 for success or -1 for error.
GetEditMode()	int GetEditMode() Returns the edit mode of the current file. Returns 0 if the edit mode is Insert, or 1 if the edit mode is Typeover. Returns -1 if a file is not open.
GetEnv()	int GetEnv(string str1, string str2) Fills 'str1' with the content of the environment variable named in 'str2'. Returns 1 for success or -1 for error.
GetFloat()	int GetFloat(string prompt, float x) Displays a message box with 'prompt' and fills 'x' with the value entered by the user. Returns 1 for success or -1 for error.
GetGMTDateTime()	int GetGMTDateTime(int y, int m, int d, int hh, int mm, int ss) Gets the current date and time at GMT (Greenwich Mean Time) and fills 'y', 'm', 'd', 'hh', 'mm' and 'ss' with year/month/day/hour/minute/second, respectively. Hours will be in 24-hour format. Returns 1 for success or -1 for error.
GetInt()	int GetInt(string prompt, int n) Displays a message box with 'prompt' and fills 'n' with the value entered by the user. Returns 1 for success or -1 for error.
GetLineText()	int GetLineText(int n, string str) Fills 'str' with the text of line 'n'. Returns the length of line 'n' or -1 for error.
GetMonName()	int GetMonName(string str, int n) Fills 'str' with the 3-character name of month number 'n' (1-12). The string returned is sensitive to the local language. Returns 1 for success or -1 for error.
GetMonthName()	int GetMonthName(string str, int n) Fills 'str' with the full name of month number 'n' (1-12). The string returned is sensitive to the local language. Returns 1 for success or -1 for error.
GetProgramDirectory()	int GetProgramDirectory(string str) Fills 'str' with the full path of the program directory. Returns 1 for success or -1 for error. See also GetDataDirectory().
GetReadOnly	int GetReadOnly Returns the read-only state of the current file. Returns 1 if the file is read-only, 0 if the file is not

	read-only, or -1 for error. See also: ToggleReadOnly()
GetSelection()	int GetSelection(string str) Fills 'str' with the currently selected text. Returns the length of the selection or -1 for error. See also: ActiveClipboard
GetSelectionBounds()	int GetSelectionBounds(int l1, int c1, int l2, int c2) Fills l1, c1, l2, c2 with the bounds of the currently selected text. Returns 1 for success or -1 for error.
GetSelectionMode()	int GetSelectionMode() Returns 0 if the current selection mode is Stream, 1 if the current selection mode is Columnar
GetSelectionSize	int GetSelectionSize Returns the number of character currently selected, 0 if a selection is not present, or -1 for error.
GetShortName()	int GetShortName(string shortname, string fullname) Fills 'shortname' with the 8.3/DOS format short filename that corresponds to 'longname'. Returns 1 for success, or -1 for error.
GetString()	int GetString(string prompt, string result [, string default]) Displays a message box with 'prompt' and fills 'result' with the string entered by the user. If the optional third parameter 'default' is present, it is suggested as the default entry string. Returns the length of 'result' or -1 for error.
GetTextWidth	int GetTextWidth Returns the current Text Width value.
GetTime12()	int GetTime12(int h, int m, int s, int pm) Gets the current time and fills 'h', 'm' and 's' with hours, minutes and seconds, respectively. Hours will be in 12-hour format. If the time is PM, 'pm' is set to 1, else it is set to 0. Returns 1 for success or -1 for error.
GetTime24()	int GetTime24(int h, int m, int s) Gets the current time and fills 'h', 'm' and 's' with hours, minutes and seconds, respectively. Hours will be in 24-hour format. Returns 1 for success or -1 for error.
GetWeekday()	int GetWeekday(int y, int m, int d) Returns the number of the weekday associated with the date 'y', 'm', 'd'. Returns 1-7 for success or -1 for error.

GetWeekdayName()	int GetWeekdayName(string str, int n) Fills 'str' with the full name of weekday number 'n' (1-7). The string returned is sensitive to the local language. Returns 1 for success or -1 for error.
GetWindowNumber()	int GetWindowNumber(string fn) Returns the window number that holds the file 'fn'. Returns -1 for error, 0 if the named file is not open, or a postive value if the file's window is located. See also Filename(), SwitchToWindow().
GetWord()	int GetWord(string str) Fills 'str' with the word at the text cursor. Returns the length of the word found or -1 for error. See also SelectWord().
GetWordDelimiters()	int GetWordDelimiters(string str) Fills 'str' with a string that contains the characters considered to be word delimiters for the current file. Returns 1 for success or -1 for error.
GetYesNo()	int GetYesNo(string title, string query) Gets a Yes or No reply from the user. Displays a message box with title 'title' and message 'query'. Returns 1 if the user clicks Yes, 0 if the user clicks No.
GoToByteOffset()	int GoToByteOffset(int n OR string str) Go to offset 'n' in the current file. Returns 1 for success or -1 for error. A string parameter is also accepted. For example: "+25" will cause the cursor to be moved ahead 25 bytes. If a parameter is not provided the Go to Byte Offset dialog will appear when the macro is run.
GoToColumn()	int GoToColumn(int n OR string str) Go to column 'n' in the current file. Returns 1 for success or -1 for error. A string parameter is also accepted. For example: "-12" will cause the cursor to be moved 12 columns to the left. If a parameter is not provided the Go to Column dialog will appear when the macro is run.
GoToLine()	int GoToLine(int n or string str) Go to line 'n' in the current file. Returns 1 for success or -1 for error. A string parameter is also accepted. For example: "+50" will cause the cursor to be moved ahead 50 lines. If a parameter is not provided the Go to Line dialog will appear when the macro is run.
GoToParagraph()	int GoToParagraph(int n) Go to paragraph 'n' in the current file. Returns 1 for

	success or -1 for error. If a parameter is not provided the Go to Paragraph dialog will appear when the macro is run.
HardenLineEnders	Issues the Harden Line Enders command.
HTMLImageTag()	int HTMLImageTag(string fn) Inserts an HTML 'IMG' tag for the image file 'fn'. BMP, GIF and JPG images are supported. Returns 1 for success or -1 for error.
Increment()	int Increment(int n) Adds 'n' to the value at the text cursor and places the result in the text file. Returns the result of the operation or -1 for error. If 'n' is not supplied the Increment dialog will appear when the macro is run.
IndentOneSpace	Issues the Indent One Space command
IndentOneTabstop	Issues the Indent One Tabstop command
IndentWithString()	int IndentWithString(string str) Indents the selected lines with 'str'. Returns 1 for success or -1 for error. If 'str' is not supplied the Indent with String dialog will appear when the macro is run.
InsertCharacter()	int InsertCharacter(char ch) Inserts character 'ch' into the edited text. Returns the ASCII value of 'ch' or -1 for error. (This command is identical to PutChar.)
InsertFile()	int InsertFile(string str) Insert file 'str' into the current file. Returns 1 for success or -1 for error. If 'str' is not provided the Insert File dialog will appear when the macro is run.
InsertFilename	Issues the Insert Filename command
InsertLineAbove	Issues the Insert Line Above command
InsertLineBelow	Issues the Insert Line Below command
InsertLongDate	Issues the Insert Long Date command
InsertLongTime	Issues the Insert Long Time command
InsertMode	InsertMode Switches the edit mode to Insert in the current file. See also ToggleEditMode().
InsertShortDate	Issues the Insert Short Date command
InsertShortTime	Issues the Insert Short Time command
InvertLines	Issues the Invert Lines command
isalnum()	int isalnum(char c)

	Returns non-zero if character 'c' is alphanumeric.
isalpha()	int isalpha(char c) Returns non-zero if character 'c' is alphabetic.
isascii()	int isascii(char c) Returns non-zero if character 'c' is in the range 0-127.
IsBookmarked()	int IsBookmarked(int n) Returns 1 if line 'n' is bookmarked, 0 if not, or -1 for error. If 'n' is not supplied, the current line is assumed.
iscntrl()	int iscntrl(char c) Returns non-zero if character 'c' is a control character (0-31 or 127).
isdigit()	int isdigit(char c) Returns non-zero if character 'c' is a digit.
islower()	int islower(char c) Returns non-zero if character 'c' is lowercase.
ispunct()	int ispunct(char c) Returns non-zero if character 'c' is punctuation.
isspace()	int isspace(char c) Returns non-zero if character 'c' is whitespace (space, tab, newline, etc.).
isupper()	int isupper(char c) Returns non-zero if character 'c' is uppercase.
isxdigit()	int isxdigit(char c) Returns non-zero if character 'c' is a hex digit (A-F, a-f, 0-9).
JustificationStyle()	int JustificationStyle(int n) Sets the current text justification style according to 'n': 1=Left, 2=Center, 3=Right, 4=Smooth. If 'n' is not supplied the Justification Style dialog will appear when the macro runs. Returns 1 for success or -1 for error.
LastCharacter()	int LastCharacter(string str) Returns the last character in 'str' or 0 if 'str' is an empty string.
Left	int Left(int n) Issues the Left command 'n' times. Returns the number of commands performed or -1 for error. The argument 'n' is optional; if it is not provided a single command is performed.
LeftWindowEdge	Issues the Left Window Edge command
LineContains()	int LineContains(int n, string str)

	Returns the number of occurrences of 'str' that appear in line 'n'. The search performed is case sensitive. Regular expressions are not recognized.
LineContainsi()	int LineContainsi(int n, string str) Returns the number of occurrences of 'str' that appear in line 'n'. The search performed is case insensitive. Regular expressions are not recognized.
LineContainsRE()	int LineContainsRE(int n, string str) Returns the number of matches to 'str' that appear in line 'n'. The search performed is case sensitive. Regular expressions ARE recognized.
LineContainsREi()	int LineContainsREi(int n, string str) Returns the number of matches to 'str' that appear in line 'n'. The search performed is case insensitive. Regular expressions ARE recognized.
LineCount	int LineCount Returns the number of lines in the current file.
LineDrawing()	int LineDrawing(int style) Initiates or terminates Line Drawing mode. If 'style' is 1 to 11, a corresponding line style from the Line Drawing dialog is activated. The Up, Down, Left and Right commands can then be used to draw lines and boxes. When 'style' is 0, Line Drawing mode is terminated. If 'style' is not supplied the Line Drawing dialog will appear when the macro runs. Returns 1 for success or -1 for error.
LineIsEmpty()	int LineIsEmpty(int n) Returns TRUE if line 'n' is empty. Note: a line containing only whitespace is considered empty.
LineLength()	int LineLength(int n) Returns the number of characters in line 'n'.
LineNumber	int LineNumber Returns the current line number in the current file or -1 for error.
LineSpacing	int LineSpacing(int mode) Formats the range of selected lines, or the whole file, according to 'mode'. 'mode' can be 1, 2 or 3, which produces single, double or triple spacing, respectively. Returns 1 for success or -1 for error.
log()	float log(float x) Returns the natural log of 'x'. 'x' must be a positive value greater than 0.
log10()	float log(float x) Returns the base 10 log of 'x'. 'x' must be a positive value greater than 0.

MakeLineBottom	Issues the Make Line Bottom command
MakeLineCenter	Issues the Make Line Center command
MakeLineTop	Issues the Make Line Top command
max()	int max(int n1, int n2) Returns the greater of 'n1' and 'n2'.
Maximize	Maximize the current editing window.
MaximizeAll	Issues the Maximize All command
Message()	Message(string str, ...) Displays a pop-up message box with title 'str' and a message that is built from all arguments that follow. Example: Message("Results", n, " removed;", m, " remain.");
min()	int min(int n1, int n2) Returns the lesser of 'n1' and 'n2'.
Minimize	Minimize the current editing window.
MinimizeAll	Issues the Minimize All command
Modified	int Modified Returns 1 if the current file has been modified, else 0. Returns -1 for error. See also SetModified().
MoveLineDown	Issues the Move Line Down command
MoveLineUp	Issues the Move Line Up command
Multiply()	int Multiply(int n) Multiplies the value at the text cursor by 'n' and places the result in the text file. Returns the result of the operation or -1 for error. If 'n' is not supplied the Multiply dialog will appear when the macro is run.
New	Issues the New command
NextBookmark	Issues the Next Bookmark command
NextFunction	Issues the Next Function command
NextParagraph	Issues the Next Paragraph command
OEMChart	Issues the OEM Chart command
OEMtoANSI	Issues the OEM to ANSI command
Open()	int Open(string fn) Opens the file 'fn' for editing. If 'fn' is already open for editing, its window will become the current window. Returns 1 for success or -1 for error. Remember: \\ must be used to denote \ within 'fn'.

OpenEmail()	int OpenEmail(string str) Initiates an email message to the address in 'str' using the default email client. Returns 1 for success or -1 for error. See also OpenEmailAtCursor.
OpenEmailAtCursor	Issues the Open Email at Cursor command
OpenFileInBrowser	Issues the Open File in Browser command
OpenFilenameAtCursor	Issues the Open Filename at Cursor command
OpenHeaderFile	Issues the Open Header File command
OpenHex()	int OpenHex(string fn) Opens the file 'fn' for hex mode viewing and editing. Returns 1 for success or -1 for error. Remember: \\ must be used to denote \ within 'fn'.
OpenProgramAtCursor	Issues the Open Program at Cursor command
OpenRecentFile()	OpenRecentFile(int n) Opens recent file number 'n'. When a sufficient file history exists, 'n' can range from 1 to 24.
OpenRecentProject()	OpenRecentProject(int n) Opens recent project number 'n'. When a sufficient project history exists, 'n' can range from 1 to 16.
OpenSystemFiles	Issues the Open System Files command
OpenURL()	int OpenURL(string str) Opens the URL described in 'str' in the default internet browser. Returns 1 for succes or -1 for error. See also OpenURLAtCursor.
OpenURLAtCursor	Issues the Open URL at Cursor command
PageDown	Issues the Page Down command
PageLeft	Issues the Page Left command
PageRight	Issues the Page Right command
PageSetup	Issues the Page Setup command
PageUp	Issues the Page Up command
Paste	Issues the Paste command
PasteAs	Issues the Paste As command
PasteClipboard()	PasteClipboard(int n) Pastes the content of Clipboard 'n' into the current file. The Windows Clipboard is number 0; private clipboards are numbered 1 to 8.

Pause	Pause Pauses macro execution by displaying a message box and waiting for it to be closed.
pi	float pi Returns the value of pi, which is approximately 3.1415926536.
PlaySound()	int PlaySound(string filepath) Plays the .WAV file described in 'filepath'. Returns 1 for success or -1 for error.
pow()	float pow(float x, float y) Returns the value of 'x' raised to the power 'y'.
PressChar()	int PressChar(string prompt, char c) Displays the message 'prompt' on the status bar and fills 'c' with the next character pressed by the user. A popup dialog does NOT appear. A file must be open in order for PressChar to operate. Returns 1 for success or -1 for error. Note: PressChar will not wait for a character when run in Debug mode. See also GetChar().
PreviousBookmark	Issues the Previous Bookmark command
PreviousFunction	Issues the Previous Function command
PreviousParagraph	Issues the Previous Paragraph command
Print	Issues the Print command
PrintAll	Issues the Print All command
PrintAllColor	Issues the Print All Color command
PrintAllMonochrome	Issues the Print All Monochrome command
PrintColor	Issues the Print Color command
printf()	int printf(string format, ...) Processes 'format' and inserts a string into the edited text, in accordance with the formatting commands used in 'C'. Returns the number of characters inserted. See the online help for more information.
PrintMonochrome	Issues the Print Monochrome command
PrintPreview	Issues the Print Preview command
PrintPreviewColor	Issues the Print Preview Color command
PrintPreviewMonochrome	Issues the Print Preview Monochrome command
PrintSetup	Issues the Print Setup command

ProjectAddAll	Issues the Project Add All command
ProjectAddOne	Issues the Project Add One command
ProjectAutoUpdate()	int ProjectAutoUpdate(int mode) Toggles the Auto-Update feature on or off for the active project according to 'mode'. Returns 1 for success or -1 for error.
ProjectClose	Issues the Project Close command
ProjectDelete()	int ProjectDelete(string name) Deletes the project file described by 'name'. A confirmation prompt will be presented. Returns 1 for success or -1 for error.
ProjectEditActive	Issues the Project Edit Active command
ProjectEditOther()	int ProjectEditOther(string name) Opens the project file described by 'name' for editing. If 'name' does not exist an empty file will be opened. Returns 1 for success or -1 for error.
ProjectName()	int ProjectName(string fn) Fills 'fn' with the full path of the active project file. Returns the length of the filepath or -1 for error.
ProjectNew	Issues the Project New command
ProjectOpen()	int ProjectOpen(string name) Open the project file described by 'name'. Returns 1 for success or -1 for error.
ProjectRemove	Issues the Project Remove command
ProjectUpdateAll	Issues the Project Update All command
ProjectUpdateOne	Issues the Project Update One command
PutChar()	int PutChar(char ch) Inserts character 'ch' into the edited text. Returns 1 for success or -1 for error.
PutClipboardText()	int PutClipboardText(string str, int n) Fills Clipboard 'n' with string 'str'. Returns the length of the text installed or -1 for error. The Windows Clipboard is number 0; private clipboards are numbered 1 to 8. See also AppendToClipboard().
PutFloat()	int PutFloat(float x) Inserts the value of 'x' into the edited text. Two decimal places will be used. Returns 1 for success or -1 for error. Use printf() if special formatting is required.
PutInt()	int PutInt(int n)

	Inserts the value of 'n' into the edited text. Returns 1 for success or -1 for error.
PutLineText()	int PutLineText(int n, string str) Replaces the text of line 'n' with 'str'. Returns the length of 'str' or -1 for error.
PutMany()	int PutMany(...) Inserts the supplied argument(s) into the edited text. Example: PutMany(5, " is a number", '\n');
PutSelection()	int PutSelection(string str) Inserts string 'str' into the edited text. Returns the length of 'str' or -1 for error. PutSelection() is the complement to GetSelection(), and it should be used instead of PutString to ensure proper insertion of column-selected text.
PutString()	int PutString(string str) Inserts string 'str' into the edited text. Returns the length of 'str' or -1 for error.
PutWordDelimiters()	int PutWordDelimiters(string str) Sets the word delimiters for the current file to the characters contained in 'str'. Returns 1 for success or -1 for error.
QuoteAndReformat	Issues the Quote and Reformat command
Random()	int Random(int n) Returns a random number between 0 and n-1 or -1 for error.
ReadValue()	int ReadValue(string name, char/int/string/float val) Reads a value from the macro variable storage area named 'name' and places it into variable 'val'. The type of 'val' must agree with the type used when the value was written using WriteValue(). Returns 1 for success or -1 for error. See also WriteValue(), EraseValue(), ValueExists().
Redo	Issues the Redo command
RedoAll	Issues the Redo All command
Reference	Issues the Reference command
Reformat	Issues the Reformat command
ReloadFile	Issues the Reload File command
RenameFile()	int RenameFile(string oldname, string newname) Renames the file or directory named 'oldname' to 'newname'. Files can be renamed across drives; directories must be on the same drive. The target 'newname' must not exist. Returns 1 for success or

	-1 for error.
Replace()	int Replace(string str1, string str2) Searches for 'str1' and replaces it with 'str2'. Returns the number of replacements made or -1 for error. The user will be prompted to confirm replacements. If 'str1' and 'str2' are not supplied the Replace dialog will appear when the macro is run. Replace() will use the current settings on the Replace dialog, but it will force the Perl Regular Expressions option OFF, and the Match Case option ON. See also Replacei().
ReplaceAgain	Issues the Replace Again command
ReplaceAll()	int ReplaceAll(string str1, string str2) Searches for 'str1' and replaces it with 'str2'. Returns the number of replacements made or -1 for error. The user will NOT be prompted to confirm replacements. ReplaceAll() will use the current settings on the Replace dialog, but it will force the Perl Regular Expressions option OFF, and the Match Case option ON. See also ReplaceAlli().
ReplaceAlli()	int ReplaceAlli(string str1, string str2) Searches for 'str1' and replaces it with 'str2'. Returns the number of replacements made or -1 for error. The user will NOT be prompted to confirm replacements. ReplaceAlli() will use the current settings on the Replace dialog, but it will force the Perl Regular Expressions option OFF, and the the Match Case option OFF. See also ReplaceAll().
ReplaceAllRE()	int ReplaceAllRE(string str1, string str2) Searches for 'str1' and replaces it with 'str2'. Returns the number of replacements made or -1 for error. The user will be prompted to confirm replacements. If 'str1' and 'str2' are not supplied the Replace dialog will appear when the macro is run. ReplaceAllRE() will use the current settings on the Replace dialog, but it will force the Perl Regular Expressions option to ON, and the Match Case option ON. See also ReplaceAll(), ReplaceAlli and ReplaceAllREi().
ReplaceAllREi()	int ReplaceAllREi(string str1, string str2)

	Searches for 'str1' and replaces it with 'str2'. Returns the number of replacements made or -1 for error. The user will be prompted to confirm replacements. If 'str1' and 'str2' are not supplied the Replace dialog will appear when the macro is run. ReplaceAllREi() will use the current settings on the Replace dialog, but it will force the Perl Regular Expressions option to ON, and the Match Case option OFF. See also ReplaceAll(), ReplaceAlli and ReplaceAllRE().
Replacei()	int Replacei(string str1, string str2) Searches for 'str1' and replaces it with 'str2'. Returns the number of replacements made or -1 for error. The user will be prompted to confirm replacements. If 'str1' and 'str2' are not supplied the Replace dialog will appear when the macro is run. Replacei() will use the current settings on the Replace dialog, but it will force the Perl Regular Expressions option OFF, and the Match Case option OFF. See also Replace().
ReplaceLineEnders()	int ReplaceLineEnders(string str1, string str2) Searches for 'str1' and replaces it with 'str2'. 'str1' and 'str2' may use the sequence \n (if within a quoted string) to represent a line ender. Returns the number of replacements made or -1 for error. The user will be NOT prompted to confirm replacements. If 'str1' and 'str2' are not supplied the Replace Line Enders dialog will appear when the macro is run. ReplaceLineEnders() will use the current settings on the Replace Line Enders dialog.
ReplaceRE()	int ReplaceRE(string str1, string str2) Searches for 'str1' and replaces it with 'str2'. Returns the number of replacements made or -1 for error. The user will be prompted to confirm replacements. If 'str1' and 'str2' are not supplied the Replace dialog will appear when the macro is run. ReplaceRE() will use the current settings on the Replace dialog, but it will force the Perl Regular Expressions option to ON, and the Match Case option ON. See also Replace(), Replacei() and ReplaceREi().
ReplaceREi()	int ReplaceREi(string str1, string str2) Searches for 'str1' and replaces it with 'str2'. Returns the number of replacements made or -1 for

	error. The user will be prompted to confirm replacements. If 'str1' and 'str2' are not supplied the Replace dialog will appear when the macro is run. ReplaceREi() will use the current settings on the Replace dialog, but it will force the Perl Regular Expressions option to ON, and the Match Case option OFF. See also Replace(), Replacei() and ReplaceRE().
Restore	Restore the current window from a minimized or maximized state.
RestoreAll	Issues the Restore All command
RestoreSettings	RestoreSettings Restores a variety of editor settings which were earlier noted using SaveSettings(). These functions can be used to ensure that a macro does not alter the editor's settings. The following settings are restored: Wordrwrap, Text Width, Justification Style, Edit Mode, Tab Display Size, Selection Mode, Active Clipboard and Word Delimiters.
Right	int Right(int n) Issues the Right command 'n' times. Returns the number of commands performed or -1 for error. The argument 'n' is optional; if it is not provided a single command is performed.
RightWindowEdge	Issues the Right Window Edge command
ROT5	Applies a ROT5 (rotation 5) conversion to the selected text
ROT13	Applies a ROT13 (rotation 13) conversion to the selected text
ROT18	Applies a ROT18 (rotation 18) conversion to the selected text
ROT47	Applies a ROT47 (rotation 47) conversion to the selected text
Round()	float Round(float x, int n) Returns the value of 'x' rounded to 'n' decimal places.
RunProgram()	int RunProgram(string fn, string params, string workdir, int wait) Runs the named program, document or folder in 'fn' by passing it to the ShellExecuteEx Windows API call. Command line parameters can be passed in 'params'. The program's working directory can be passed in 'workdir'. If 'wait' is 1, macro execution will be suspended until the program has been

	closed. Returns the completion code of ShellExecuteEx: non-zero for success, zero for error. See also OpenProgramAtCursor().
Save	Issues the Save command
SaveACopyAs()	int SaveACopyAs(string fn) Saves a copy of the current file to the file 'fn'. The name of the current file is not changed. Returns 1 for success or -1 for error. Remember: \\ must be used to denote \ within 'fn'.
SaveAll	Issues the Save All command
SaveAs()	int SaveAs(string fn) Saves the current file to the file 'fn'. The name of the current file is changed to 'fn'. Returns 1 for success or -1 for error. Remember: \\ must be used to denote \ within 'fn'.
SaveSelectionAs()	int SaveSelectionAs(string fn) Saves the current selection to the file 'fn'. Returns 1 for success or -1 for error. Remember: \\ must be used to denote \ within 'fn'.
SaveSettings	SaveSettings Records a variety of editor settings for later restoration using RestoreSettings(). These functions can be used to ensure that a macro does not alter the editor's settings. The following settings are saved: Wordrwrap, Text Width, Justification Style, Edit Mode, Tab Display Size, Selection Mode, Active Clipboard and Word Delimiters.
ScrollDown	Issues the Scroll Down command
ScrollLeft	Issues the Scroll Left command
ScrollRight	Issues the Scroll Right command
ScrollUp	Issues the Scroll Up command
SelectAllText	Issues the Select All Text command
SelectColumnar	Issues the Select Columnar command
SelectDown	Issues the Select Down command
SelectLeft	Issues the Select Left command
SelectPageDown	Issues the Select Page Down command
SelectPageLeft	Issues the Select Page Left command
SelectPageRight	Issues the Select Page Right command
SelectPageUp	Issues the Select Page Up command
SelectRight	Issues the Select Right command

SelectStream	Issues the Select Stream command
SelectToBottomOfPage	Issues the Select to Bottom of Page command
SelectToEndOfFile	Issues the Select to End of File command
SelectToEndOfLine	Issues the Select to End of Line command
SelectToStartOfFile	Issues the Select to Start of File command
SelectToStartOfLine	Issues the Select to Start of Line command
SelectToTopOfPage	Issues the Select to Top of Page command
SelectUp	Issues the Select Up command
SelectWithoutShift	Issues the Select without Shift command
SelectWord	int SelectWord(string str) Selects the word at the text cursor and places it in 'str'. Returns the length of the word selected or 0 if no word could be found to select. See also GetWord().
SelectWordLeft	Issues the Select Word Left command
SelectWordRight	Issues the Select Word Right command
SetClipboard()	SetClipboard(int n) Sets the active clipboard to Clipboard 'n'. The Windows Clipboard is number 0; private clipboards are numbered 1 to 8.
SetClipboardNext	Issues the Set Clipboard Next command
SetClipboardPrevious	Issues the Set Clipboard Previous command
SetCurrentDirectory	int SetCurrentDirectory(string str) Sets the current directory for the active process to 'str'. Returns 1 for success or -1 for error. See also GetCurrentDirectory().
SetModified	SetModified Sets the modifed state of the current to true. This might be used to force a Save operation even when a file has not been modified. Returns 1 for success or -1 for error. See also Modified().
ShadedTabZones()	int ShadedTabZones(int mode) Enables or disables the Shaded Tab Zones display mode according to 'mode'.
sin()	float sin(float x) Returns the sine of 'x'. The angle 'x' must be in radians.

sinh()	float sinh(float x) Returns the hyperbolic sine of 'x'. The angle 'x' must be in radians.
SoftenLineEnders	Issues the Soften Line Enders command.
SortFileTabsByExt()	int SortFileTabsByExt(int mode) Enables or disables the sorting of File Tabs by extension according to 'mode'. If 'mode' is 1 the feature is enabled. If 'mode' is 0 the feature is disabled.
SortFileTabsByName()	int SortFileTabsByName(int mode) Enables or disables the sorting of File Tabs by name according to 'mode'. If 'mode' is 1 the feature is enabled. If 'mode' is 0 the feature is disabled.
SortFileTabsByUse()	int SortFileTabsByUse(int mode) Enables or disables the sorting of File Tabs by name according to 'mode'. If 'mode' is 1 the feature is enabled. If 'mode' is 0 the feature is disabled.
SortLines	Issues the Sort Lines command
Space	Issues the Space command. If a range of lines is selected, the range will be indented.
SpacesToTabs	Issues the Spaces to Tabs command
SpellChecker	Issues the Spell Checker command
SplitHorizontal	Issues the Split Horizontal command
SplitVertical	Issues the Split Vertical command
sprintf()	int sprintf(string str1, string format, ...) Processes 'format' and builds an output string in 'str1', in accordance with the formatting commands used in 'C'. Returns the length of 'str1' or -1 for error. See the online help for more information.
sqrt()	float sqrt(float x) Returns the positive square root of 'x'.
StartOfFile	Issues the Start of File command
StartOfLine	Issues the Start of Line command
StatusMessage()	StatusMessage(...) Displays a message in the status bar that is built from all arguments that follow. Example: StatusMessage(n, " were removed;", m, " remain.");
strcat()	int strcat(string str1, string str2) Concatenates 'str2' to 'str1'. Returns the length of 'str1' or -1 for error.
strchr()	int strchr(string str, char c) Returns the offset at which the character 'c' appears

	in 'str' or -1 if 'c' does not appear. See also strrchr().
strcmp()	int strcmp(string str1, string str2) Compares 'str1' to 'str2' with case sensitivity. Returns 0 if the strings are equal. Returns < 0 if 'str1' is less than 'str2'. Returns > 0 if 'str1' is greater than 'str2'.
strcmpi()	int strcmpi(string str1, string str2) Compares 'str1' to 'str2' without case sensitivity. Returns 0 if the strings are equal. Returns < 0 if 'str1' is less than 'str2'. Returns > 0 if 'str1' is greater than 'str2'.
strcpy()	int strcpy(string str1, string str2) Copies 'str2' to 'str1'. Returns the length of 'str1' or -1 for error.
StripHTMLTags	Issues the Strip HTML/XML Tags command
StripLeadingSpaces	Issues the Strip Leading Spaces command
StripTrailingSpaces	Issues the Strip Trailing Spaces command
strlen()	int strlen(string str) Returns the length of 'str'.
strlwr()	int strlwr(string str) Converts the string 'str' to lowercase. Returns the length of 'str'.
strncat()	int strncat(string str1, string str2, int n) Concatenates up to 'n' characters from 'str2' to 'str1'. Returns the length of 'str1' or -1 for error. A NULL byte is added after the characters added.
strncmp()	int strncmp(string str1, string str2, int n) Compares up to 'n' characters in 'str1' to 'str2' with case sensitivity. Returns 0 if the strings are equal. Returns < 0 if 'str1' is less than 'str2'. Returns > 0 if 'str1' is greater than 'str2'.
strncmpi()	int strncmpi(string str1, string str2, int n) Compares up to 'n' characters in 'str1' to 'str2' without case sensitivity. Returns 0 if the strings are equal. Returns < 0 if 'str1' is less than 'str2'. Returns > 0 if 'str1' is greater than 'str2'.
strncpy()	int strncpy(string str1, string str2, int n) Copies up to 'n' characters from 'str2' to 'str1'. Returns the length of 'str1' or -1 for error. A NULL byte is added after the characters copied.
strrchr()	int strrchr(string str, char c) Returns the offset at which the character 'c' last

	appears in 'str' or -1 if 'c' does not appear. See also strchr().
strrev()	int strrev(string str) Reverses the string 'str' in place. Example: The string 'Boxer' would be converted to 'rexoB'.
strstr()	int strstr(string str1, string str2) Searches string 'str1' for the substring 'str2' with case sensitivity. Returns the offset at which 'str2' is found or -1 if not found.
strstri()	int strstri(string str1, string str2) Searches string 'str1' for the substring 'str2' without case sensitivity. Returns the offset at which 'str2' is found or -1 if not found.
strupr()	int strupr(string str) Converts the string 'str' to uppercase. Returns the length of 'str'.
SubString()	int SubString(string str1, string str2, int index, int len) Fills 'str1' with up to 'len' characters from 'str2', starting at offset 'index'. The first character in a string is referred to by offset 0. Returns the new length of 'str1' or -1 for error. If 'index' is negative, and 'len' is positive, 'str1' will be filled with 'len' characters starting 'index' characters in from the end of 'str2'. If 'len' is negative, the value of 'index' is ignored, and 'str1' is filled with the rightmost 'len' characters from 'str2'.
SwapLines	Issues the Swap Lines command
SwapWords	Issues the Swap Words command
SwitchToWindow()	SwitchToWindow(int n) Makes window 'n' active. Windows are numbered from 1 to the number of open files. See also FileCount. See also GetWindowNumber().
SyntaxHighlightAs	Issues the Syntax Highlight As command
Tab	Issues the Tab command. If a range of lines is selected, the range will be indented.
TabDisplaySize()	int TabDisplaySize(string str) Sets the Tab Display Size for the current file

	according to 'str'. If a single value appears in 'str', tabs are set to fixed width with the value in 'str'. If a series of comma-separated values are found in 'str', variable width tab stops will be used. Returns TRUE for success, -1 for error. If 'str' is not supplied the Tab Display Size dialog will appear when the macro is run.
TabsToSpaces	Issues the Tabs to Spaces command
tan()	float tan(float x) Returns the tangent of 'x'. The angle 'x' must be in radians.
tanh()	float tanh(float x) Returns the hyperbolic tangent of 'x'. The angle 'x' must be in radians.
Templates	Issues the Templates command
TextIsSelected	int TextIsSelected Returns 1 if a stream selection is present, 2 if a columnar selection is present, or 0 if no selection is present. Returns -1 for error.
TextWidth()	int TextWidth(int n) Sets the Text Width to 'n'. Returns TRUE for success or -1 for error. If 'n' is not supplied the Text Width dialog will appear when the macro is run.
TileAcross	Issues the Tile Across command
TileDown	Issues the Tile Down command
ToggleBookmark()	int ToggleBookmark(int n, int state) Sets bookmark 'n' according to 'state'. Returns TRUE for success, -1 for error. If 'state' is ON, bookmark 'n' is placed on the current line. If 'state' is OFF bookmark 'n' is cleared, where ever it is.
ToggleEditMode()	ToggleEditMode(int state) Toggles the edit mode according to 'state'. If 'state' is 1, Insert mode is used. If 'state' is 0, Typeover mode is used. See also the functions InsertMode() and TypeoverMode().
ToggleReadOnly()	ToggleReadOnly(int state) Toggle read-only mode according to 'state'. If 'state' is 1, read-only mode is set. If 'state' is 0, read-only mode is released.
tolower()	int tolower(char c) Returns the lowercase mate to character 'c'.
ToolbarBottom	Issues the Toolbar Bottom command

ToolbarLeft	Issues the Toolbar Left command
ToolbarRight	Issues the Toolbar Right command
ToolbarTop	Issues the Toolbar Top command
TopLine	int TopLine Returns the line number of the first line in the editor window. Returns -1 for error.
TopOfPage	Issues the Top of Page command
TotalAndAverage	Issues the Total and Average command
TouchFile()	int TouchFile(string name) Touch (ie, update the timestamp of) the file named 'name'. Returns 1 for success or -1 for error.
toupper()	int toupper(char c) Returns the uppercase mate to character 'c'.
Trim()	int Trim(string str) Removes leading and trailing blanks from 'str'. Returns the new length of 'str'.
TrimLeft()	int TrimLeft(string str) Removes leading blanks from 'str'. Returns the new length of 'str'.
TrimRight()	int TrimRight(string str) Removes trailing blanks from 'str'. Returns the new length of 'str'.
Trunc()	float Trunc(float x, int n) Returns the value of 'x' truncated to 'n' decimal places.
TypeoverMode	TypeoverMode Switches the edit mode to Typeover in the current file. See also ToggleEditMode().
Uncomment	Issues the Uncomment command.
Undo	Issues the Undo command.
UndoAll	Issues the Undo All command.
UndoAllClosedTabs	Issues the Undo All Closed Tabs command.
UndoClosedTab()	UndoClosedTab(int n) Reopens recently closed file tab number 'n'. When a sufficient closed tab list exists, 'n' can range from 1 to 10.
UndoCloseTab	Issues the Undo Close Tab command.
Unformat	Issues the Unformat command.

UnformatXML	Issues the Unformat XML command.
UnhighlightMatches	Issues the Unhighlight Matches command.
Unindent	Issues the Unindent command.
Up	int Up(int n) Issues the Up command 'n' times. Returns the number of commands performed or -1 for error. The argument 'n' is optional; if it is not provided a single command is performed.
UserList()	UserList(int n) Opens User List window 'n'.
UserTool()	UserTool(int n) Runs User Tool number 'n'.
UserToolWait()	UserTool(int n) Runs User Tool number 'n' and waits for it to complete execution.
ValueAtCursor	int ValueAtCursor Returns the ASCII value of the character at the cursor, or -1 if a character is not available at the cursor.
ValueExists()	int ValueExists(string name) Looks for the variable 'name' in the macro variable storage area. Returns 1 if it exists, 0 if it does not exists, or -1 for error. See also ReadValue(), WriteValue(), EraseValue().
ViewBookmarks()	int ViewBookmarks(int mode) Enables or disables the viewing of Bookmarks according to 'mode'.
ViewFileTabs()	int ViewFileTabs(int mode) Enables or disables the viewing of File Tabs according to 'mode'.
ViewHexMode()	int ViewHexMode(int mode) Enables or disables read-only hex mode viewing according to 'mode'.
ViewHexRuler()	int ViewHexRuler(int mode) Enables or disables the viewing of the Hex Ruler according to 'mode'.
ViewTextRuler()	int ViewTextRuler(int mode) Enables or disables the viewing of the Text Ruler according to 'mode'.
ViewHScrollBar()	int ViewHScrollBar(int mode) Enables or disables the viewing of Horizontal Scroll Bars according to 'mode'.

ViewLineNumbers()	int ViewLineNumbers(int mode) Enables or disables the viewing of Line Numbers according to 'mode'.
ViewRightMarginRule()	int ViewRightMarginRule(int mode) Enables or disables the viewing of the Right Margin Rule according to 'mode'.
ViewStatusBar()	int ViewStatusBar(int mode) Enables or disables the viewing of the Status Bar according to 'mode'.
ViewSyntaxHighlighting()	int ViewSyntaxHighlighting(int mode) Enables or disables the Syntax Highlighting feature according to 'mode'.
ViewTextHighlighting()	int ViewTextHighlighting(int mode) Enables or disables the Text Highlighting feature according to 'mode'.
ViewToolbar()	int ViewToolbar(int mode) Enables or disables the viewing of the Toolbar according to 'mode'.
ViewVisibleSpaces()	int ViewVisibleSpaces(int mode) Enables or disables the viewing of Visible Spaces according to 'mode'.
ViewVScrollBar()	int ViewVScrollBar(int mode) Enables or disables the viewing of Vertical Scroll Bars according to 'mode'.
VisualWrap()	int VisualWrap(int mode) Enables or disables Visual Wrap according to 'mode'.
VisualWrapOptions	Issues the Visual Wrap Options command.
Wait()	int Wait(int n) Delays macro execution for 'n' milliseconds. 1000 milliseconds equals 1 second. Returns 1 for success or -1 for error.
WindowHeight	int WindowHeight Returns the number of lines that can be displayed in the current window.
WindowLastVisited	Issues the Window Last Visited command
WindowList	Issues the Window List command
WindowNext	Issues the Window Next command
WindowPrevious	Issues the Window Previous command
WindowSkip	int WindowSkip(int mode) If 'mode' is 1, sets the skip status for the current window to on. If mode is 0, skip status is turned off. Returns 1 for success or -1 for error.

WindowWidth	int WindowWidth Returns the number of columns that can be displayed in the current window.
WordCount()	int WordCount(int lines, int words, int chars) Fills the supplied variables with the count of lines, words and characters, respectively. Returns 1 for success or -1 for error.
WordLeft	int WordLeft(int n) Issues the Word Left command 'n' times. Returns the number of commands performed or -1 for error. The argument 'n' is optional; if it is not provided a single command is performed.
WordRight	int WordRight(int n) Issues the Word Right command 'n' times. Returns the number of commands performed or -1 for error. The argument 'n' is optional; if it is not provided a single command is performed.
WordWrap()	int WordWrap(int mode) Enables or disables Typing Wrap according to 'mode'. Deprecated: see TypingWrap().
WriteValue()	int WriteValue(string name, char/int/string/float val) Writes 'val' to the macro variable storage area named 'name'. 'name' will be visible to other macros, so be careful to choose a unique identifier. Returns 1 for success or -1 for error. See also ReadValue(), EraseValue(), ValueExists().
xtoi()	int xtoi(string str) Returns the integer value of the hexadecimal number described by string 'str'. Returns -1 for error.

5.158 Macro Language Reference

Data Types

Boxer's Macro Language supports the following data types:

```
string
char
int
float
```

A `string` can hold a series of characters up to 2,048 bytes in length. The end of a string is marked with a Null character (ASCII 0). A string constant is enclosed within double quotes.

The `char` data type is an 8-bit, unsigned data type which can hold values in the range 0 to 255. A character constant is enclosed within single quotes.

The `int` data type is a 32-bit, signed data type which can hold integer values in the range -2,147,483,648 to 2,147,483,647.

The `float` data type is a double precision, signed data type that can hold values in the range 2.2250738585072014e-308 to 1.7976931348623158e+308.

Keywords

The following words are reserved keywords and may not be used as variable names:

```
break       int         true
continue    char        false
do          string      yes
else        float       no
for         void        on
goto                    off
if
macro
return
while
```

The keywords listed above are case sensitive, and must be entered in lowercase. The symbolic constants in the third column (`true, false, yes, no, on, off`) are an exception: they can appear in lowercase, uppercase, or even in mixed case.

Arithmetic Operators

The following arithmetic operators are supported:

Operator	Meaning
+	addition
-	subtraction
*	multiplication
/	division
%	modulus
++	increment
--	decrement

The modulus operator (`%`) returns the remainder from an integer division operation. For example, the expression `n = 7 % 4` will result in `n` receiving the value `3`, since `7 / 4` leaves a remainder of `3`.

The increment and decrement operators can be used to increase or decrease an integer

variable by 1. The expression:

```
i++;
```

is equivalent to:

```
i = i + 1;
```

The ++ and -- operators can be used in either prefix or postfix location. If i has an initial value of 3, the statement:

```
n = i++;
```

will leave n with the value of 3, while i is incremented to 4. The incrementing of i occurs *after* the assignment due to the postfix location.

Assuming i again starts with a value of 3, the statement:

```
n = --i;
```

will leave n with a value of 2 and i with a value of 2. The decrementing of i occurs *before* the assignment due to the prefix location.

The addition (+) operator has been overloaded to support string concatenation. The following statements:

```
string s1 = "Boxer ";
string s2 = "Text Editor";
string s3 = s1 + s2;
```

would result in s3 having the value: "Boxer Text Editor"

Assignment Operators

The following assignment operators are supported:

Operator	Meaning
=	assignment
+=	addition assignment
-=	subtraction assignment
*=	multiplication assignment
/=	division assignment
%=	modulus assignment
&=	bitwise AND assignment
|=	bitwise OR assignment

^=	bitwise XOR assignment
<<=	left shift assignment
>>=	right shift assignment

The assignment operator (=) should be familiar to all. The other operators which each conclude with = all represent a shorthand notation. For example, the statement:

```
i += 5;
```

is equivalent to:

```
i = i + 5;
```

The += operator has been *overloaded* to support string concatenation. The following statements:

```
string str = "Boxer ";
str += "Text Editor";
```

would result in `str` having the value: `"Boxer Text Editor"`

The last five operators listed above are bitwise assignment operators. Their function is analogous to the += operator; see the Bitwise Operators section of this topic for some additional detail.

Boolean Operators

The following Boolean operators are supported:

Operator	Meaning
==	equal
!=	not equal
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
&&	logical AND
||	logical OR
!	logical NOT (unary negation)
~=	case insensitive string comparison

The operators ==, !=, <, >, <= and >= have been overloaded to allow operations on strings. A string is considered greater than another string if it would appear higher

in an alphabetic sort. In other words, the statement:

```
if ("apple" < "zebra")
```

evaluates to TRUE.

The first nine operators above are standard to most high-level languages. The last operator is specific to Boxer's Macro Language, and permits strings to be compared without case sensitivity. For example, the statement:

```
if ("MasterCard" ~= "mastercard")
```

would evaluate to TRUE.

Bitwise Operators

The following bitwise operators are supported:

Operator	Meaning
&	bitwise AND
|	bitwise OR
^	bitwise XOR
<<	left shift
>>	right shift
~	one's complement (unary)

A full discussion of bitwise arithmetic would be beyond the scope of this language reference. For those who are interested, any introductory book on the C programming language would be a suitable reference. The information below will be sufficient to remind those with prior experience of the function of each operator:

& Sets a bit to 1 in the result if and only if both of the corresponding bits in its operands are 1, and to 0 if the bits differ or both are 0. Example: 9 & 1 yields 1.

| Sets a bit to 1 in the result if one or both of the corresponding bits in its operands are 1, and to 0 if both of the corresponding bits are 0. Example: 9 | 2 yields 11.

^ Sets a bit in the result to 1 when the corresponding bits in its operands are different, and to 0 when they are the same. Example: 7 ^ 4 yields 3;

<< Shifts the first operand the number of bits to the left specified in the second operand, filling with zeros from the right. Example: 2 << 3 yields 16.

>> Shifts the first operand the number of bits to the right specified in the second operand, discarding the bits that 'fall off' at the right. Example: 34 >> 2 yields 8.

~ Inverts each bit in the operand, changing all ones to zeros and all zeros to ones. Example: `~0xFFFF0000` yields `0x0000FFFF`.


The large majority of users will never find a need for bitwise arithmetic, but it has been included in the interest of completeness.

Operator Precedence

The following table summarizes operator precedence and order of evaluation for the various operators supported by Boxer's Macro Language. Operators with the strongest/highest precedence are listed first:

Operator	Evaluates
( ) [ ]	left to right
! ~ ++ -- -	right to left
* / %	left to right
+ -	left to right
<< >>	left to right
< <= > >=	left to right
== != ~=	left to right
|	left to right
&	left to right
^	left to right
&&	left to right
||	left to right
? :	right to left
= += -= etc.	right to left
,	left to right

Parentheses can be used when required to ensure that the order of evaluation occurs as desired. For example:

```
n1 = 3 * 5 + 4;
```

assigns `19` to `n1`, while:

```
n1 = 3 * (5 + 4);
```

assigns `27` to `n1`.


Because the assignment operator (=) is evaluated from right to left, a construction such as the following is possible:

```
int i, j, k;
i = j = k = 0;
```

k is assigned the value 0, j is assigned the value of k, and i is assigned the value of j.

Character Constants

Boxer's Macro Language recognizes the standard character constants which have been popularized by the C programming language:

Sequence	Meaning	Decimal Value
'\b'	Backspace	8
'\f'	Formfeed	12
'\n'	Newline	10
'\r'	Carriage Return	13
'\t'	Tab	9
'\\'	Backslash	92
'\''	Single Quote	39
'\"'	Double Quote	34
'\0'	Null	0

In addition, Boxer will recognize a backslash (\) followed by three *octal* digits as the character whose ASCII value is given by the digits used. For example, '\101' could be used to represent a capital A, since its ASCII value, in octal, is 101.

Character constants can be used in any place that a char data type is expected, or within a double-quoted string: "this is a string with a newline at the end.\n"

Numeric Constants

Numeric int constants can be specified in either *decimal* or *hexadecimal* format:

```
int n1 = 32;
int n2 = 0x20;
```

Each of these assignments supplies the value 32 to n1 or n2.

Numeric float constants can be specified in any of the following forms:

```
float x1 = 500;
float x2 = 500.0;
float x3 = 5e2;
```

```
float x4 = 5e02;
float x5 = 5.0e2;
float x6 = 5.0e02;
float x7 = 5.0e+2;
float x7 = 5.0e+02;
```

Each of these assignments results in the value 500 being assigned to the variable being declared.

For floating point values less than 1, the minus sign can be used to designate exponentiation. All of the following examples represent the number .05:

```
.05
0.05
5e-2
5e-02
5.0e-2
5.0e-02
```

Symbolic Constants

The following symbolic constants are recognized:

Name	Value
TRUE	1
FALSE	0
YES	1
NO	0
ON	1
OFF	0

These constants can be used in place of the values 0 and 1 to make a macro more readable. For example, you can write:

```
ViewBookmarks(ON);
```

instead of:

```
ViewBookmarks(1);
```

Declaring Variables

Variable names can be up to 32 characters in length and must not conflict with the names of any keywords or internal functions. Variable names can use alphanumeric characters and the underscore (_), but they must not start with a digit. All variables must be declared before use. Initialization of variables can be done at declaration-time, but this is not required. Uninitialized variables will be zero-filled automatically.

Boxer's Macro Language supports a flexible syntax for declaring variables. All of the following examples are legal declarations when they appear at the top of a macro, before other executable statements:

```
string s1;
string s2 = "Boxer";
string s3, s4, s5;
string s6 = "abc", s7, s8 = "def";

char c1;
char c2 = 'A';
char c3, c4, c5;
char c6, c7 = 'x', c8;

int n1;
int n2 = 10;
int n3, n4, n5;
int n6, n7 = -4, n8;

float x1;
float x2 = 1.05;
float x3 = 1.2e04;
float x4, x5, x6;
float x7, x8 = 7.75, x9;
```

In the spirit of the C programming language, Boxer's macro language also allows a `string` variable to be declared as an array of characters. The declaration:

```
char str[100];
```

is (for most purposes) functionally equivalent to the declaration:

```
string str;
```

for declaring a variable which can hold a short string of characters. See the *String Subscripting* section below for details on when the former style might be required.

Conditional Statements

Boxer's Macro Language supports three different conditional statements: `if`, `if-else` and the ternary statement. An `if` statement will be executed if the expression in parentheses evaluates to a non-zero result. Below are examples of the three conditional statements:

```
if (LineCount() > 10000)
   {
   longfile = true;
   }


if (LineCount() > 10000)
   {
```

```
   longfile = true;
   }
else
   {
   longfile = false;
   }


longfile = (LineCount() > 10000) ? true : false;
```

In the first example, the variable `longfile` is set `TRUE` if the return from the function `LineCount()` is greater than `10000`. In the second example, an `if-else` statement is used to additionally set `longfile` to `FALSE` if the condition is *not* met.

The final example illustrates the ternary statement, and its effect is identical to the `if-else` example immediately above it. If the condition within parentheses evaluates to `TRUE`, the expression immediately following the `?` is evaluated. If not, the expression after the `:` is evaluated. A ternary statement is effectively a compact `if-else` statement.


The ternary statement in Boxer's Macro Language is modeled after that of the C programming language, with one exception. In Boxer macros, the parentheses around the conditional expression are *required*, in C these parentheses are optional.


When a single statement is conditional upon an `if` or `if-else` statement, as is shown in the examples above, the use of curly braces `{ }` is not required. Curly braces *are* required when two or more statements are to be conditionally executed, or when those statements are the subject of a looping statement.

Looping Statements

Boxer's Macro Language supports three different looping statements: `for`, `while` and `do-while`. A loop statement will continue looping so long as the 'test' expression in parentheses evaluates to a non-zero result. Below are examples of each of these statements:

```
// find the longest line in the file
for (line = 1, longest = 0; line <= LineCount(); line++)
   if ((n = LineLength(line)) > longest)
         longest = n;


// find the longest line in the file
line = 1;
longest = 0;
while (line <= LineCount())
   {
   if ((n = LineLength(line)) > longest)
         longest = n;
```

```
   line++;
   }


// find the longest line in the file
line = 1;
longest = 0;
do
   {
   if ((n = LineLength(line)) > longest)
        longest = n;

   line++;
   }
while (line < LineCount());
```

The three loops above are functionally equivalent to one another, with one exception that will be discussed below.

The `for` loop is the most compact, since it permits the three elements of a loop's control to be specified on a single line: the initialization, the test, and the increment. These are found within the parentheses of the `for` loop and are separated by semi-colons. When a `for` loop is first executed, the initialization section is performed, and the test section is evaluated. If the test evaluates to a non-zero result, the statement(s) in the body of the loop are processed. At the end of the loop, the increment section is processed. Control then passes again to to the test section, to the body, and so on.


Boxer's Macro Language supports a very flexible `for` loop structure. The initialization, test and increment sections are each optional. Moreover, multiple initializations can be performed by separating the statements with the comma operator.

The `while` loop is a simpler loop, in that the only required control element that must be supplied is the test. For illustration purposes, the `while` loop above was written to be identical in function to the `for` loop above it. In fact, every `for` loop can be written as a `while` loop, and every `while` loop can be written as a `for` loop. A `for` loop is typically used when one needs to initialize and increment a loop index. A `while` loop is typically used when a single condition is sufficient to control the flow of the loop.

A `do-while` loop is essentially an upside-down `while` loop. A `do-while` loop tests at the bottom, whereas a `while` loop tests at the top. A `do-while` loop should be used in those cases where the loop is always to be be executed at least once. That leads us to why the `do-while` example above is not exactly equivalent to the `for` and `while` loops above it. If the current file is empty, the `for` and `while` loops above will not be executed. The `LineCount()` function will return `0` and the initial test will fail. In the `do-while` loop, the `LineCount()` call isn't made until the bottom of the loop. In the case of an empty file, the body of the loop would be processed and the `LineLength()` call would fail because the `line` parameter would

be out of range.

Sometimes the need arises to construct a 'forever' loop; one which will run until some condition within the body of the loop is satisfied and a `break` statement is executed. Both the `for` and `while` loops can be used for this purpose. Here are two examples:

```
// loop until the user enters the right answer
for (;;)
   {
   GetString("What's the capital of Arizona?", answer);

   if (answer ~= "Phoenix")
        break;
   }

// loop until the user enters the right answer
while (TRUE)
   {
   GetString("What's the capital of New Hampshire?", answer);

   if (answer ~= "Concord")
        break;
   }
```

Notice that these examples used the `~=` operator to ensure that the user's response was not rejected due to improper case.

Alert readers might notice that the above examples could be more neatly implemented using a `do-while` loop, since this is a case where the loop always wants to be run once, and the test can be more logically placed at the bottom of the loop:

```
do
   GetString("What's the capital of California?", answer);
while (strcmpi(answer, "Sacramento") != 0);
```

This example uses the `strcmpi()` function to perform a case insensitive string comparison, because the `~=` operator does not have a companion *string-does-not-match* operator.

The break Statement

The `break` statement can be used to exit from a loop prematurely. Control passes to the next statement following the loop which has been exited. For example:

```
// loop on all lines in the file
for (i = 1; i <= LineCount(); i++)
   {
   // exit the loop if a line is longer than 1000 characters
   if (LineLength(i) > 1000)
        break;
```

```
   }

// control passes to here after break
New;
```

The continue Statement

The `continue` statement can be used to jump to the bottom of a loop prematurely. Control passes to an imaginary label at the end of the loop. For example:

```
// loop on all lines in the file
for (i = 1; i <= LineCount(); i++)
   {
   // exit the loop if a line is longer than 1000 characters
   if (LineLength(i) > 1000)
        continue;

   // ... other processing ...

   // continue jumps to here
   }
```

The goto Statement

The `goto` statement can be used to jump unconditionally to a label. Control passes to the next statement after the label. For example:

```
// loop on all lines in the file
for (i = 1; i <= LineCount(); i++)
   {
   // exit the loop if a line is longer than 1000 characters
   if (LineLength(i) > 1000)
        goto toolong;

   // ... other processing ...

   toolong:
   // goto jumps to here

   // ... other processing ...
   }
```

The return Statement

The `return` statement can be used to end a macro prematurely. If a return statement is not encountered, a macro will run until the closing curly brace in the body of the macro is encountered.

Function Calls

Boxer's Macro Language includes a wide variety of functions that provide access to the editor's commands, configuration settings, and to string and math libraries. The

function set is documented in the Macro Function Reference, as well as in the Macro Dialog itself.

When making a function call, care should be taken to ensure that the parameters supplied to the function match the declared type(s) that the function expects to receive. Boxer is able to trap missing and/or mismatched parameters in most cases, but unexpected results can occur when invalid parameters are supplied.

Function names are not case sensitive; Boxer will accept function names that do not match the function name with regard to character case.

If a function does not require parameters, it is not necessary to supply parentheses at the end of the function name. For example:

```
LineCount();
```

and

```
LineCount;
```

are functionally equivalent, because the `LineCount` function does not require any parameters. That said, the practice of using `()` on all function calls can help to distinguish function names from variable names.

Simple expressions can be supplied to in a function call without difficulty, and they will be evaluated as expected before being sent to the function for processing. For example:

```
max(3 * 45, 4 * 90);
```

is a legitimate construction that might be used in calling the `max()` function. If you find that you are getting unexpected results in a case like this, introduce a temporary variable to hold the value of the expression, and then supply the variable to the function in place of the expression.

String Subscripting

Arrays are not supported in the classical sense; it's not possible to declare an array of `int` or `float` variables, for example. But Boxer's Macro Language does recognize a `string` variable to be an array of elements of type `char`, and allows those elements to be accessed individually through the use of subscripts. The first character within a string is located at index `0`, the second character is at index `1`, etc. In the following example:

```
string str = "BOXER";
char c1;
c1 = str[2];
```

the character variable `c1` would be assigned the value `'X'`.

Likewise, a `string` variable can be modified by assigning individual elements within

the string using subscripting:

```
string s1 = "water";
s1[0] = 'w';
s1[1] = 'i';
s1[2] = 'n';
s1[3] = 'e';
s1[4] = '\0';
```

This code fragment has the effect of changing the content of string variable `s1` from `"water"` to `"wine"`. Notice that the null character (`'\0'`) was used to shorten the string from five characters to four.

String subscripting makes it possible to use a string variable in the way that an array might be used. Here's an example that totals the number of occurrences of each letter within an input string:

```
macro array_example()
{
int i;
string input = "now is the time for all good men to come to the
aid of their country.";
char tally[256];        // note that all elements are initially
set to zero

// loop to process all characters in the input string
for (i = 0; input[i] != '\0'; i++)
   tally[input[i]]++;

// open a new, untitled file
New;

// report the results for lowercase letters
for (i = 'a'; i <= 'z'; i++)
   printf("letter %c occurred %d time(s)\n", i, tally[i]);
}
```

Had the `tally` array been declared as a `string` type, Boxer's built-in range checking would have prevented the string from being used in the way that was shown above. By declaring the string as a character array of sufficient size, the macro processor is forewarned that the code may later index into the string beyond the terminating null character.

Due to the capacity of the `char` data type (0-255), the utility of the above technique is limited to applications in which the maximum number of occurrences would be less than 256.

Type Conversions

Boxer's Macro Language will automatically convert between data types whenever possible in order to resolve an expression that involves mismatched data types. Here are some examples:

```
   string s1 = 'A';          // result: s1 gets "A" (char to string)
   string s2 = 65;           // result: s2 gets "A" (int to string)
   string s3 = 65.0;         // result: s3 gets "A" (float to string)

   char c1 = 65;             // result: c1 gets 'A' (int to char)
   char c2 = "A";            // result: c2 gets 'A' (string to char)
   char c3 = 65.0;           // result: c3 gets 'A' (float to char)

   int n1 = 'A';             // result: n1 gets 65; (char to int)
   int n2 = "123";           // result: n2 gets 123 (string to int)
   int n3 = 123.45;          // result: n3 gets 123 (float to int)

   float x1 = 'A'            // result: x1 gets 65.0 (char to float)
   float x2 = 65;            // result: x2 gets 65.0 (int to float)
   float x3 = "123.45";      // result: x3 gets 123.45 (string to
float)
```

Comments

Comments can be placed throughout a macro to help document the code. Two types of comments are supported, block comments and end-of-line comments:

```
/* this is a multi-line
   block comment */

int n1 = 7;       // this is an end-of-line comment
```

5.159 Make Line Bottom

Menu: Jump > Make Line Bottom

Default Shortcut Key: none

Macro function: MakeLineBottom()

The Make Line Bottom command causes the screen to be redrawn so that the current line is at screen bottom. This command is useful for showing the text above the current view without losing the current line.

If there is insufficient text to fill the window, the command will be disabled.

5.160 Make Line Center

Menu: Jump > Make Line Center

Default Shortcut Key: none

Macro function: MakeLineCenter()

The Make Line Center command causes the screen to be redrawn so that the current line is at the middle of the screen.

5.161 Make Line Top

Menu: Jump > Make Line Top

Default Shortcut Key: none

Macro function: MakeLineTop()

The Make Line Top command causes the screen to be redrawn so that the current line is at the top of screen. This command is useful for showing the text below the current view without losing the current line.

If there is insufficient text to fill the window, the command will be disabled.

5.162 Maximize All

Menu: Window > Maximize All

Default Shortcut Key: none

Macro function: MaximizeAll()

The Maximize All command can be used to maximize all editor windows. The current window will consume the entire *client area*, and all other windows can be conceptualized as having been placed behind the current window.


When an editor window is maximized, its Minimize/Maximize/Close icon set is moved to the far right of the main menu bar. Once the window is minimized or restored, its icons are moved back to its title bar.

5.163 Minimize All

Menu: Window > Minimize All

Default Shortcut Key: none

Macro function: MinimizeAll()

The Minimize All command can be used to quickly minimize all open editor windows. Minimized windows are placed at the bottom of the *client area* in iconic form. The size and position of each window is stored so that any window can assume its former position once restored. Windows can be restored individually or with the Restore All

command. To neatly arrange minimized icons which have become out of line, use the Arrange Icons command.

5.164 Move Line Down

Menu: Edit > Line > Move Line Down

Default Shortcut Key: Shift+Alt+Down

Macro function: MoveLineDown()

The Move Line Down command moves the text of the current line to the line below, and moves the text cursor to that line so it follows the line being moved. This command can be useful when rearranging items in an ordered list, since it removes the need to select, cut and then paste text from the clipboard as a means of reordering a series of lines.

If the current line is the last line in the file, the command has no effect.

The effect of this command is similar to that of the Swap Lines command, which used to reside in the Edit menu, and remains accessible via key assignment and its macro function.

See also: Move Line Up

5.165 Move Line Up

Menu: Edit > Line > Move Line Up

Default Shortcut Key: Shift+Alt+Up

Macro function: MoveLineUp()

The Move Line Up command moves the text of the current line to the line above, and moves the text cursor to that line so it follows the line being moved. This command can be useful when rearranging items in an ordered list, since it removes the need to select, cut and then paste text from the clipboard as a means of reordering a series of lines.

If the current line is the first line in the file, the command has no effect.

See also: Move Line Down

5.166 Multiply

Menu: Edit > Math > Multiply

Default Shortcut Key: none

Macro function: Multiply()

The Multiply command can be used to multiply an integer value (ie, a whole number, not a floating point value) at the text cursor by another integer value. A dialog box will appear to retrieve the value to multiply by. After clicking 'OK' the arithmetic is performed, and the old value is replaced by the result.

If the text cursor is situated on a character rather than a numeric value, the character value of the character at the cursor will be multiplied by the supplied value and the resultant character will be displayed. If the resultant character value is out of range, an error message will be given.

5.167 New (File)

Menu: File > New

Default Shortcut Key: Ctrl+N

Macro function: New()

The New command is used to create a new file for editing. The first new file opened in the edit session will be given the name `untitled.001`, and successive new files will use correspondingly higher file extensions to ensure uniqueness of the filename.

When a new file is first saved with either the Save or Save As command, a dialog box will appear so that a permanent name can be supplied for the file.

The size and position of the newly created window depends upon the nature of other open windows within Boxer. If other editing windows are maximized, the new window will also be created in maximized mode and will obscure the other windows below it. Otherwise, the new window will be created in 'normal' mode, and its size and position will be determined automatically by Windows.

Untitled files are not added to the Recent Files, and will not be reopened if an edit session is later restored.

You can request that new windows always be created in maximized mode with an option on the Configure | Preferences | Display options page. This option is titled *Auto-maximize new windows when created*.


If the desktop area within Boxer's main window is double-clicked, an empty/new file will be opened.

5.168 New (Project)

Menu: Project > New

Default Shortcut Key: none

Macro function: ProjectNew()

Boxer's Project feature provides a means of simultaneously loading a collection of files, and restoring those files to their previous editing states. The Project | New command creates a project file containing the names and editing options of all files that are currently open.

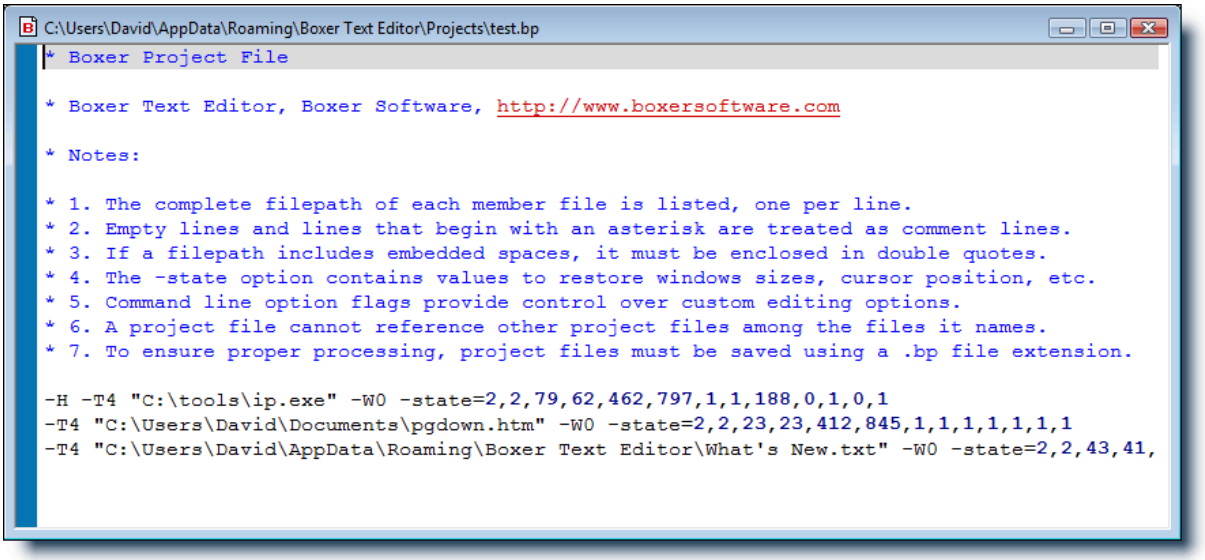
The following editing options are maintained within a project file:

- window sizes and positions
- cursor location
- active file
- bookmarks
- tab stops
- typing wrap mode
- hex editing mode
- file tab arrangement

Once a project is open, the Add One, Add All and Remove commands can be used to add and remove files from the active project. The Update One and Update All commands can be used keep the project file up-to-date with the current editing options. To keep the editing options up-to-date automatically, use the Auto-Update command.

When a project file is named on Boxer's command line, or when the icon of a project file is dragged and dropped onto the Boxer window (or its icon), all of the files named within that file will be loaded for editing. If you need to edit the content of the project file itself, use the Edit Active or Edit Other command, as may be appropriate.

The project file is a text file that contains an informative series of comments that explains the use of project files:



In its simplest form, a project file is simply a text file whose file extension is .BP and whose content consists of a list of filenames, one per line. Empty lines can be used freely within a project file to separate filenames as may be appropriate. Lines beginning with an asterisk (*) will be considered comment lines, and will not be processed.

A project file can be used to maintain a list of filenames that relates to a given project or document set, and to open those files quickly. For best results, the full pathname--including the drive designator and directory path--should be used. This will ensure that the project file functions properly regardless of the default directory in force at the time it is used.


FTP filepaths can be placed within project files. See the FTP Open command for more information.


Project files cannot be nested; if a project file is named within another project file an error will occur.


If a file is open for read-only editing, that file's entry in the project file will be automatically created with the -R command line option flag that designates read-only status. Likewise, if a file is open for hex mode viewing, its entry will be created with the -H command line option flag.


Within a project file, filepaths can be preceded with "exec:" to indicate that they be opened using their default application. This allows other files that are associated with a project to be opened when the project opens in Boxer. For example, a project file's entries might be:

```
c:\myproject\source\main.cpp
exec:c:\myproject\docs\updates.doc
exec:http://www.mysite.com/index.htm
exec:c:\myproject\bitmaps\project_logo.bmp
```

The first file would open in Boxer, while the next three files would be opened by the applications that are associated with their respective file types: DOC/HTM/BMP. If the filepath to be opened contains embedded spaces, the entire line must be surrounded in double quotes:

"exec:c:\my project\docs\monthly updates.doc"


A project file can be designated on Boxer's command line using the -P command line option flag.

5.169 Next Bookmark

Menu: Jump > Next Bookmark

Default Shortcut Key: Shift+Ctrl+Down

Macro function: NextBookmark()

The Next Bookmark command moves the text cursor to the nearest bookmark which appears below the text cursor's current location. If the Next Bookmark command finds no bookmarks below the current line, the text cursor will wrap around and be placed on the first bookmark in the file.

Travel among bookmarks is based upon location, not bookmark number.

The Bookmark Manager can be used to view all bookmarked lines in a single view, and navigate to, or delete, selected bookmarks.

Bookmarks will persist for the current editing session, and will be restored when restoring an edit session.


If a selection exists when this command is issued, the selection will be extended to the bookmarked location.

5.170 Next Function

Menu: Jump > Next Function

Default Shortcut Key: Ctrl+Alt+Down

Macro function: NextFunction()

The Next Function command moves the cursor to the next function (or procedure) declaration within the current file. This command makes it possible to move through a source code file on a function-by-function basis.


If a text selection is present when this command is issued, the selection will be extended to the new cursor location.


The Next Function command relies upon the Ctags Function Index feature to perform its service. If the Ctags feature has been disabled, or if the file being edited is not supported by Ctags, the Next Function command will be unavailable.


The types of Ctags identifiers for which this command applies is user-configurable. The default setting includes entries for "function", "procedure", "subroutine", "method", etc. The full list can be viewed or changed on the *Advanced* tab of the Configure | Ctags Function Indexing dialog.

5.171 Next Paragraph

Menu: Jump > Next Paragraph

Default Shortcut Key: none

Macro function: NextParagraph()

The Next Paragraph command moves the text cursor to the start of the next paragraph.

For purposes of this command, a paragraph is considered to be a block of lines with one or more empty lines between them. Contiguous paragraphs which are denoted by a change of indent on the first line, and not by an intervening blank line, will not be recognized to be distinct paragraphs.

If a text selection is present when this command is issued, the selection will be extended to the new cursor location.

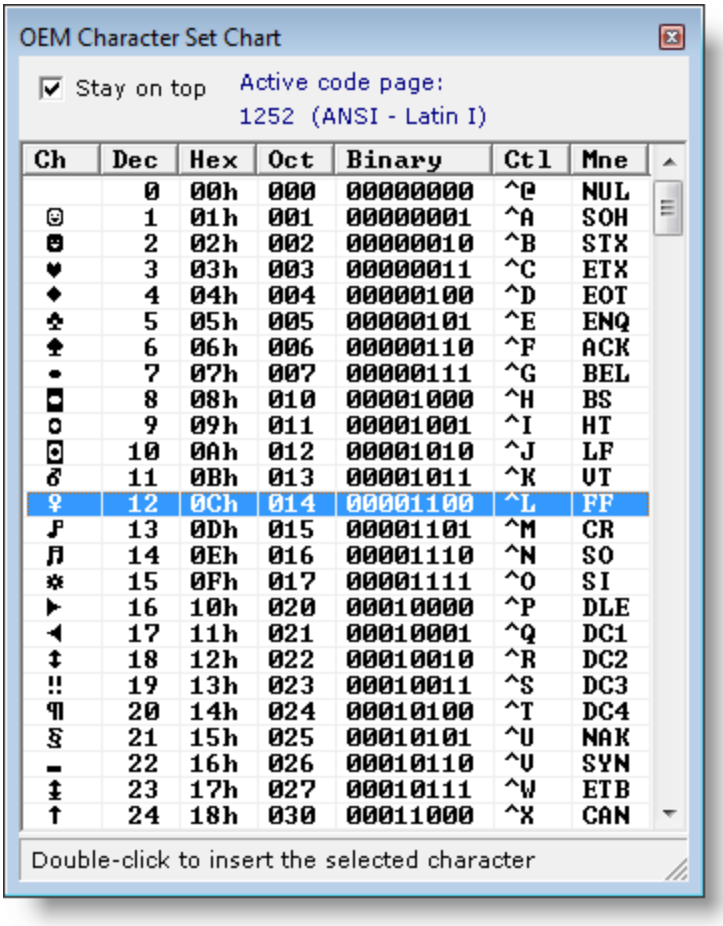
5.172 OEM Chart

Menu: Tools > OEM Chart

Default Shortcut Key: none

Macro function: OEMChart()

The OEM Chart command provides access to a popup chart which displays characters 0 to 255 in the OEM (ASCII) character set. The character's visual representation is shown in the leftmost column, followed by the character value in *decimal*, *hexadecimal*, *octal* and *binary* formats. In the two rightmost columns, the equivalent control letter sequence and mnemonic expression are shown for values in the range 0 to 31. The active code page is also displayed at the top of the dialog:



Ch	Dec	Hex	Oct	Binary	Ctl	Mne
	0	00h	000	00000000	^@	NUL
☺	1	01h	001	00000001	^A	SOH
☻	2	02h	002	00000010	^B	STX
♥	3	03h	003	00000011	^C	ETX
♦	4	04h	004	00000100	^D	EOT
♣	5	05h	005	00000101	^E	ENQ
♠	6	06h	006	00000110	^F	ACK
•	7	07h	007	00000111	^G	BEL
◘	8	08h	010	00001000	^H	BS
○	9	09h	011	00001001	^I	HT
◙	10	0Ah	012	00001010	^J	LF
♂	11	0Bh	013	00001011	^K	VT
♀	12	0Ch	014	00001100	^L	FF
♪	13	0Dh	015	00001101	^M	CR
♫	14	0Eh	016	00001110	^N	SO
☼	15	0Fh	017	00001111	^O	SI
►	16	10h	020	00010000	^P	DLE
◄	17	11h	021	00010001	^Q	DC1
↕	18	12h	022	00010010	^R	DC2
‼	19	13h	023	00010011	^S	DC3
¶	20	14h	024	00010100	^T	DC4
§	21	15h	025	00010101	^U	NAK
▬	22	16h	026	00010110	^V	SYN
↨	23	17h	027	00010111	^W	ETB
↑	24	18h	030	00011000	^X	CAN

To jump directly to a character of interest simply press that character on the keyboard.

The OEM Chart can be used to insert a character into the file being edited. Simply double click on the entry for the desired character, or highlight the character in the chart and press *Enter*. When the need to insert a special character or symbol arises frequently, consider using the Insert Symbols feature rather than the OEM Chart command. The Insert Symbols feature permits a defined character to be entered using a single keystroke.

Right-clicking on a selected item summons the OEM Chart context menu. The context menu provides an option to copy the selected character to the current clipboard.

The OEM Chart can also be used to convert between bases for values in the range 0 to 255. Simply locate the value to be converted in its proper column and read the converted value from the column of the new base.

If you prefer that the OEM Chart remain atop other windows, select the *Stay on top* option. The OEM Chart is a *non-modal* window, which allows it to remain on-screen after *focus* has been returned to another editing window.

If the OEM Chart is left on-screen when Boxer is closed, it will be automatically reopened if the edit session is later restored.

5.173 OEM to ANSI

Menu: Block > Convert Other > OEM to ANSI

Default Shortcut Key: none

Macro function: OEMtoANSI()

The OEM to ANSI command converts characters within the selected text from OEM (ASCII) character encoding to ANSI character encoding. These character encoding schemes share all the common alphabetic and numeric character mappings, but differ in the area of accented and/or graphic characters. A conversion may be appropriate when a file which was created with a DOS program must be prepared for use with a Windows program. Note that not all characters will have equivalents in the destination character set. In such cases, a conversion will not be made for that character.

Boxer's OEM Chart and ANSI Chart commands can be useful for viewing the character assignments in each of these encoding schemes.

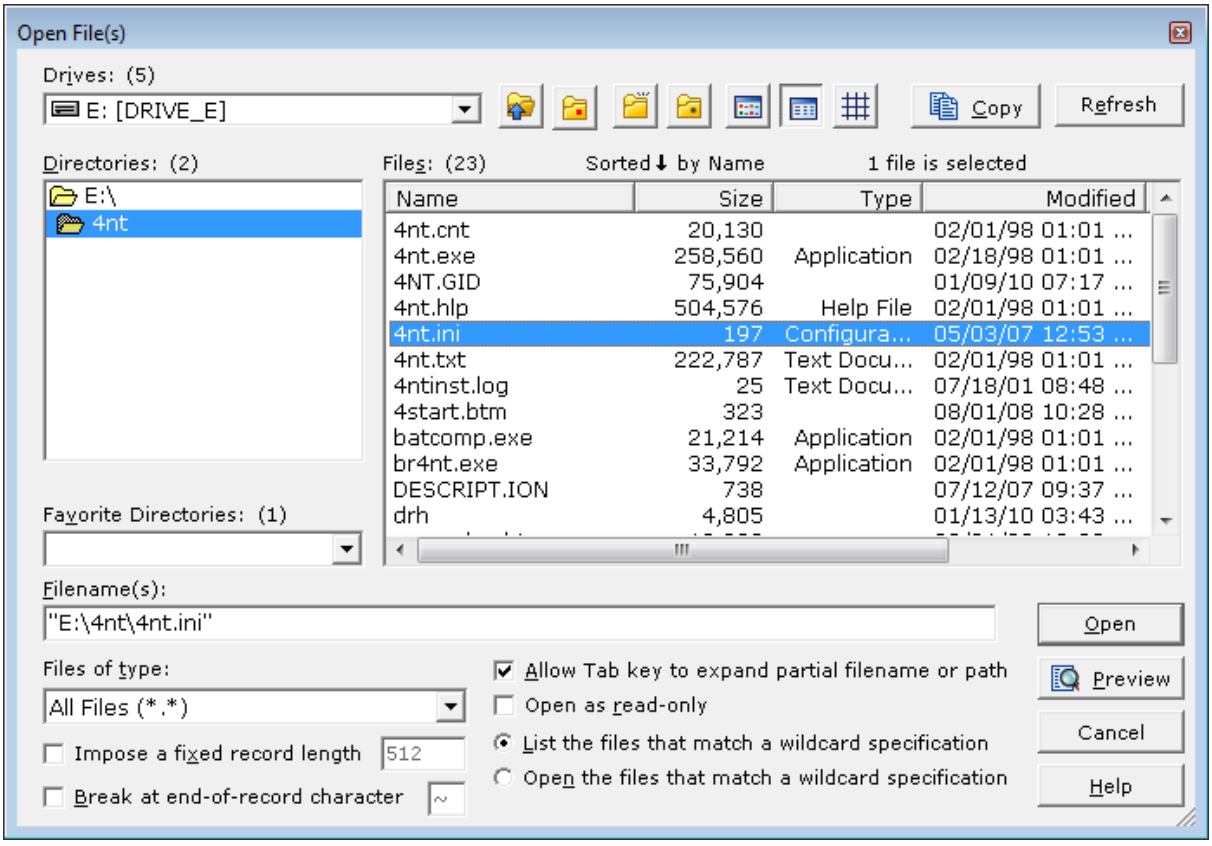
5.174 Open (File)

Menu: File > Open

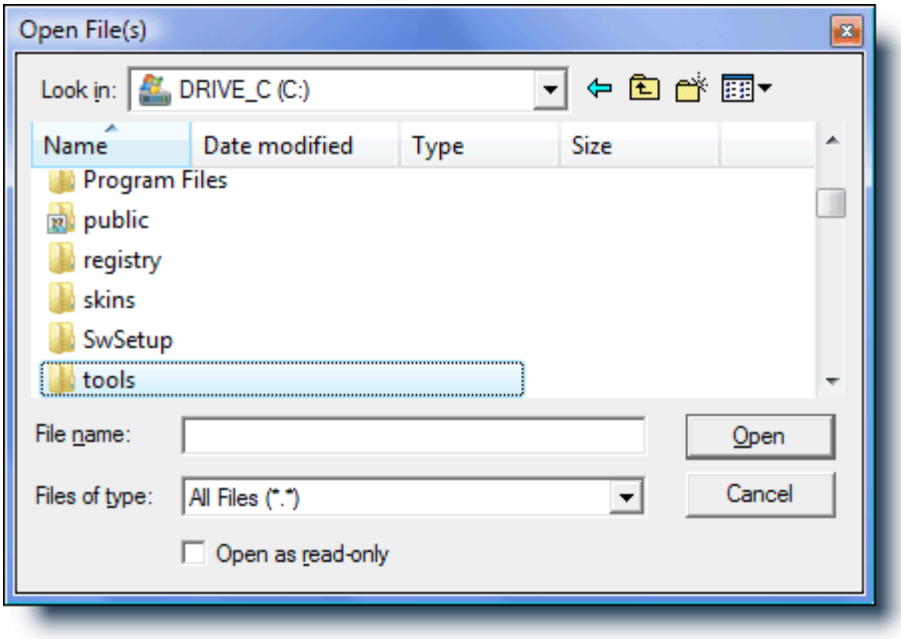
Default Shortcut Key: Ctrl+O

Macro function: Open()

The Open command is used to load one or more files for editing. By default, Boxer will present a custom File Open dialog with extra features not found in the standard Windows File Open dialog.



If you prefer to use the standard dialog, an option to do so can be found on the Configure | Preferences File I/O dialog page.



Standard Dialog or Custom Dialog

Opening a Single File
To select a single file for editing, click on the file's name within the display or type a filename in the *Filename* edit box. Click *Open* or press *Enter* to complete the selection.

Opening Multiple Files
To select multiple files for editing, depress the *Ctrl* key while clicking on a filename to add the new filename to the set of files to be opened. To select a series of adjacent files for editing, click on the first file in the series, and then depress the *Shift* key while clicking on the last file in the series.

Filtering Files
You can filter the files displayed by selecting a *file filter* from the drop-down list labeled *Files of type*. The file types which appear in this dialog are user-definable via the Configure | Preferences | File I/O options page. You can also filter the filenames displayed by typing a wildcard file specification (such as `m*.cpp` or `project.*`) into the *Filename* edit box.

Read-only Mode
To load a file for passive viewing, select the *Open as read-only* checkbox at the bottom of the dialog. Boxer will then prevent changes from being made to the file(s) during the edit session. This option can be used to protect against accidental changes to a file whose content needs to be viewed, but must not be altered.

Renaming Files
To rename a file, single-click once to select the item. Single-click again on the filename to place a text cursor into the filename. Type the new name for the file and press *Enter* .

Deleting Files
To delete one or more files, first select the files to be deleted. Use Shift+Click to select a range of files, and/or Ctrl+Click to select discontiguous files. When the files are selected, press the *Delete* key to delete the files.


Important Note: the standard Windows dialog does **not** request confirmation before deleting a file, but it **does** place the deleted file in the Windows Recycle Bin. Boxer's custom dialog **does** require that file deletions be confirmed, but does **not** place the deleted file(s) in the Recycle Bin.

Window Sizes
The size and position of the newly created window depends upon the nature of other open windows within Boxer. If other editing windows are maximized, the new window will also be created in maximized mode and will obscure the other windows below it. Otherwise, the new window will be created in 'normal' mode, and its size and position will be determined automatically by Windows.

You can request that new windows always be created in maximized mode with an option on the Configure | Preferences | Display options page. This option is titled *Auto-maximize new windows when created*.

If you would prefer that changes made to the current directory not be reflected in future visits to the dialog, an option is available on the Configure | Preferences | File I/O options page. The option is titled *File I/O dialogs preserve current directory; ignore prior travel.*

Custom Dialog Features

The following features are available only with Boxer's custom File Open dialog

Resize Dialog
Boxer's File Open dialog is resizeable, and its size and position are remembered from session to session.

Current Directory
Use this icon to make the selected directory the current working directory.

Jump to the directory of the current file
Use this icon to make the directory of the current file the current working directory.

List View or Details View
Use these icons to select whether files are displayed in list view (simple), or with full details. Unlike the standard Windows dialog, the view mode is remembered from the previous dialog session.

Details View with Grid Lines
Click the icon to the left of the *Copy* button to enable a display mode in which light grid lines will by used to separate file data.

Copy Directory Listing
Use the *Copy* button to copy the entire directory listing to the current clipboard.

Refresh Display
Click the Refresh button to refresh the file display. This option could be used if another program or process has changed the file listing since the dialog was first opened.

Directory Count
A message at the top of the drive selection list shows the number of directories being displayed in that list.

File Count
A message at the top of the filename display shows the number of files in the current directory listing.

Selected File Count
A message at the top of the filename display shows the number of files currently selected.

File Sorting

File information can be sorted by any field that appears at the top of the table:

Click twice on a given field to reverse the direction of the sort. The current sort mode is displayed in a message at the top of the filename display. Unlike the standard Windows dialog, the sort mode and direction are remembered from the previous dialog session.

The *Name* heading also offers the ability to sort filenames by extension. Clicking the *Name* heading in succession yields the following sort modes: down by name, up by name, down by extension, up by extension.

You can change the width of any field by dragging the divider line between fields to a new position.

When sorting by Type, the file entries will be sorted first by type and then by file extension within the type.

Short Name Field
In addition to the standard *Name*, *Size*, *Type*, *Modified* and *Attribute* fields, Boxer's custom dialog also provides a *Short Name* field that displays the equivalent short filename for each file.

Favorite Directories
When files are opened for editing, the directory name is saved in the Favorites list. Use the Favorites drop-down list to quickly select a directory from among those recently used. Once a directory is selected its position in the list is elevated to position one.

Boxer will automatically add the directory of the current file to the Favorite Directories list when the Open dialog is activated.

When an existing entry in the Favorites Directory is found not to exist it will be automatically deleted from the list.

Filename Completion
Boxer's custom dialog supports filename completion. When typing a filepath or filename in the *Filename* field, press the *Tab* key to begin cycling through those directories or filenames that match your partial entry. When you see the directory name you were typing appear, continue typing the next few letters of the next directory name in the filepath. Press *Tab* to complete, as required. If you press accidentally press *Tab* too many times, press *Shift+Tab* to cycle backwards to a previously displayed name.

The function of the *Tab* key for Filename Completion can be disabled with the checkbox entitled *Allow Tab to expand partial filename or path*. In this case, the *Tab* key can again be used to move away from the *Filename* box.

List the files that match a wildcard specification
Use this option if you prefer that files matching a wildcard specification first be shown in

the file list, and not opened directly. With this option, typing `*.XYZ` and clicking *Open* will cause the file list to be filtered to display only those files with a `.XYZ` extension.

Open the files that match a wildcard specification
Use this option if you prefer that files matching a wildcard specification be opened directly. With this option, typing `*.XYZ` and clicking *Open* will cause all files with a `.XYZ` extension to be opened for editing.


When entering a wildcard specification in the Filename box, the semi-colon (`;`) can be used to separate multiple file patterns. For example, to display (or open, depending on the above options) all files matching three different extensions, use the following syntax: `*.abc;*.def;*.ghi`

Impose a fixed record length as the file is read
This option can be used to impose a fixed-length record format on the file selected for opening. The record length is specified in the associated edit box. As the file is read, line breaks will be inserted at column 'value'.

This option is useful for viewing or editing files that contain fixed-length records. When the proper record size is entered, Boxer will display the file in a meaningful format, with one record per line.

Note that if the file is saved, line enders will be added. The File | Properties dialog has an option to request that line enders be removed when the file is saved.


The -F command line option flag can also be used to impose a fixed record length on the file being opened.


Because this option is dangerous to use on files that do not contain fixed-length records, the checkbox state and record length values are not recalled from the previous dialog session. Rather, the checkbox always reverts to the safe, unchecked state when the dialog is opened.

Break at end-of-record character
his This option provides support for editing files that use an end-of-record character, such as ANSI X12 files. When this option is active, Boxer will watch for the designated end-of-record character as the file is read, after which a conventional line ender will be added. By default, the inserted line enders *will* be written to the file when it is saved. To override this behavior, use the File Properties dialog to request that line enders be removed at File-Save time.


The -E command line option flag can also be used to activate this option for the next-named file on the command line.

File Preview
The *Preview* button enables you to view the content of a file without actually opening the file for editing. A pop-up window will appear that displays the opening content of the file. From this view you can decide if the file in question is the one you would like to open. The pop-up Preview window disappears automatically as soon as the mouse cursor is moved significantly.

Binary files are not eligible for display with the Preview function.

Context Menu
Right-clicking on a selected file (or files) will present the File Open dialog's context menu. The following commands are provided: Open, Preview, Select All, Copy, Rename and Delete. The Rename and Delete commands provide an alternative method to perform these functions, which are described in the section above. The Copy command copies the selected file(s) to the Windows clipboard in a manner that allows them to be pasted into Explorer, effecting a file copy. Other applications may also be able to recognize the 'Explorer Copy' clipboard format and behave accordingly.

Notes

The directory list control used by the custom dialog (known to programmers as TComboBox) does not show abstract entries such as the Desktop, Network Neighborhood, Shared Drives, etc. TComboBox is limited to the display of physical drives. All of the abstract entries have a corresponding location on the physical disk drive, if you know where to look. If you would prefer to see the abstract entries, you can revert to the standard File Open dialog using an option on the Configure | Preferences File I/O dialog page.

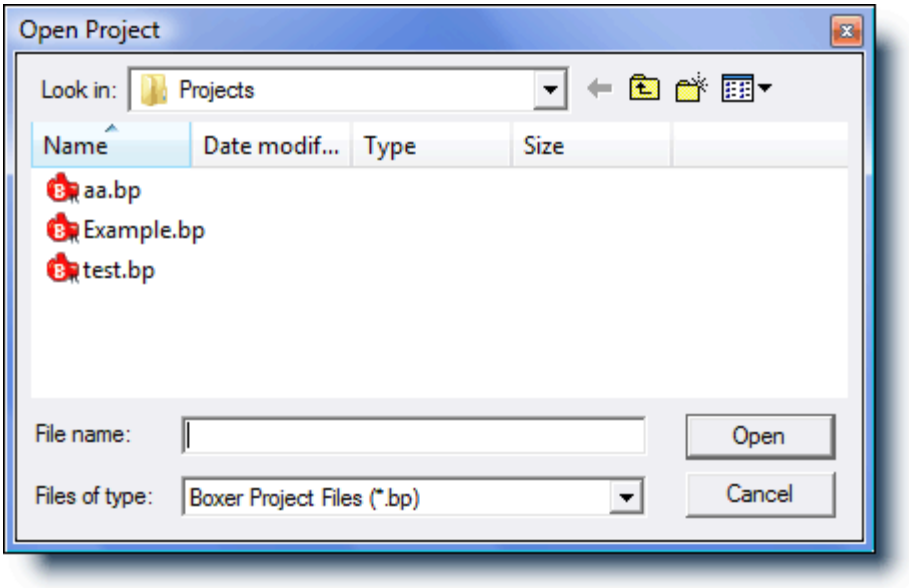
5.175 Open (Project)

Menu: Project > Open

Default Shortcut Key: none

Macro function: ProjectOpen()

The Open Project Command is used to open an existing project file. When a project file is opened, all of the files named therein are open for editing. If you need to edit the content of the project file itself, use the Edit Active or Edit Other command, as may be appropriate.



☞ See the Project | New command for full details about Boxer's project file feature.

5.176 Open Email at Cursor

Menu: File > Open Other > Email at Cursor

Default Shortcut Key: Ctrl+E

Macro function: OpenEmailAtCursor()

The Open Email at Cursor command will attempt to launch your email program to send a message to the email address beneath the text cursor. Your email program will display the email address in its '*to*' field and prompt for a subject. You can then compose your message and send it in the usual way.

This command can also be invoked by double clicking with the mouse on an email address which appears within text. The mouse cursor will change to the pointing hand when atop an email address to indicate that the address has been recognized.

☞ In order to launch your email program, Boxer relies upon the operating system shell's ability to process a 'mailto' directive. When an email client program is installed, it typically establishes itself as the program which is called by the shell to process the mailto command. If you find that your active email program is not launched by Boxer, or if some other inactive email program is launched instead, it's probably because your active email program did not establish itself to be the

program that processes mailto commands. This situation cannot be remedied by Boxer, and is not due to any shortcoming in Boxer. You might consult the documentation of your email program, or contact its vendor.

5.177 Open File in Browser

Menu: File > Open Other > File in Browser

Default Shortcut Key: Ctrl+B

Macro function: OpenFileInBrowser()

The Open File in Browser command will attempt to load and display the current file within your Internet browser program. This command is useful for reviewing the effect of changes made to HTML files, or any other files which are eligible for viewing within an Internet browser..

For those comfortable with HTML coding, this command permits Boxer to be used as a powerful HTML editor with true *WYSIWYG* display. The procedure to use is as follows: make your HTML changes within Boxer, save the file, press *Ctrl+B* to activate the browser window, and then click 'Reload' or 'Refresh' to load the latest changes from disk. Once the browser has been opened you can continue to use *Ctrl+B* from within Boxer to switch back to the browser window. Some users may find this method preferable to using a dedicated HTML editor, since many of these editors lack a true WYSIWYG display and/or comprehensive editing features.

Boxer decides whether a file is eligible for display within a browser by checking its list of eligible file extensions. This list can be edited from the Configure | Preferences | File I/O options page. The option is titled *Open File in Browser extensions*.


In order to launch your browser, Boxer relies upon the file associations which exist within the operating system between the browser and its eligible file types. Most browsers establish these file associations during their setup procedure. If you find that certain file extensions do not result in your browser being launched, it is due to the absence of the required file association(s) within the operating system, and not due to any shortcoming in Boxer itself.

5.178 Open Filename at Cursor

Menu: File > Open Other > Filename at Cursor

Default Shortcut Key: Ctrl+L

Macro function: OpenFilenameAtCursor()

Open Filename at Cursor is a timesaving command which allows the filename beneath the text cursor to be loaded for editing. Simply issue this command while the cursor is atop a filename and the file will be loaded into a new editing window. If the file does not exist, a new file with that name will be created in the current directory.

If selected text of a suitable size is present, the selected text will be used as the filename to be opened. If the filename to be opened contains spaces, it *must* be selected to ensure the full filename, including embedded spaces, will be used.

This command will be disabled when the text cursor is sitting upon a text string which could not be a valid filename, such as a series of spaces.

☞ If the filename at the cursor does not exist in the current working directory, but does exist in the directory of the currently edited file, it will be opened from that directory instead.

5.179 Open Header File

Menu: File > Open Other > Header File

Default Shortcut Key: Ctrl+H

Macro function: OpenHeaderFile()

The Open Header File command provides a method to quickly open the header file which is associated with the file being edited. The most common use of this command will be for programming, but the command's utility could be extended to any file extension pairs which are related in the same way.

Example: while editing in the C++ file `main.cpp`, issuing the Open Header File command will cause `main.hpp` to be loaded. Likewise, if `main.hpp` is being edited, issuing the Open Header File command will cause `main.cpp` to be loaded. If the associated file is already loaded within the editor, it simply becomes the active window. Issuing the command repeatedly will allow you to toggle back and forth between the two associated files.

Boxer comes with several header file associations pre-defined for common programming languages. Additional header file extension pairs can be defined on the Configure | Preferences | File I/O options page. The option is titled *Open Header File extensions*.

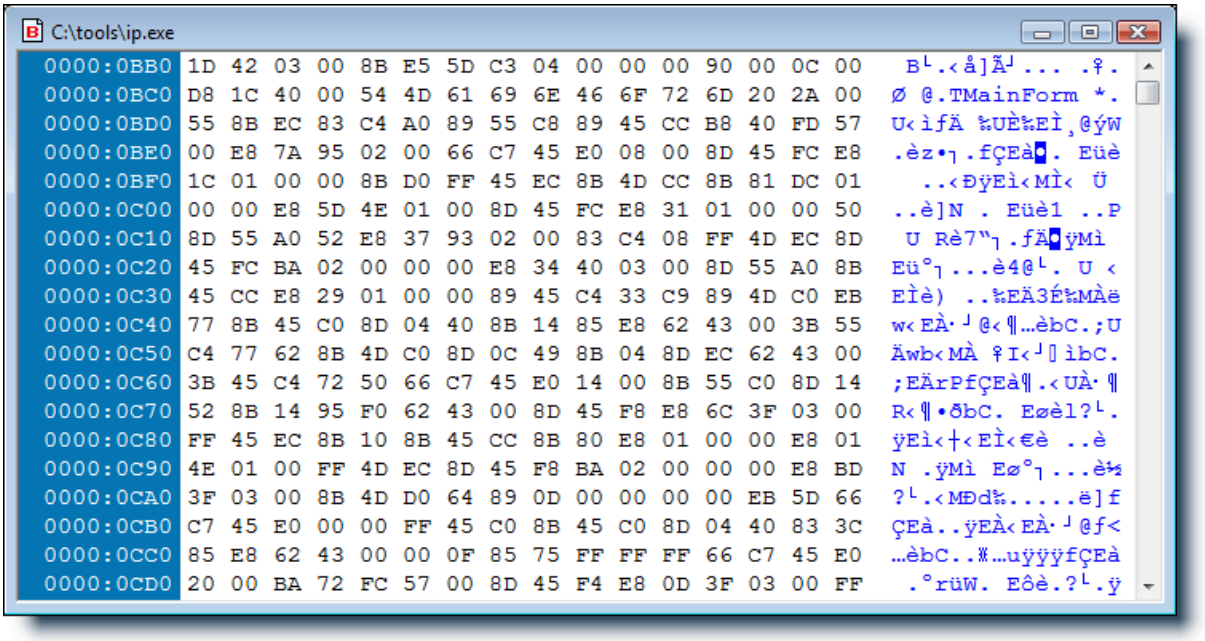
5.180 Open Hex

Menu: File > Open Hex

Default Shortcut Key: Ctrl+Alt+O

Macro function: OpenHex()

The Open Hex command is used to open a file in hex mode for viewing or editing. After selecting the file to be opened from the Open dialog, a new window is created and the file is displayed within the window:



Files opened with Open Hex are displayed in a special format which has three sections. At the left, in the line number zone, the byte offset into the file is shown in *hexadecimal* format. In the center, sixteen data bytes are displayed as two-byte hexadecimal values. At the far right the same sixteen bytes are displayed as ASCII characters, except in cases where the character in question cannot be so represented.

☞ The representation of the sixteen characters at the right depends upon whether an ANSI or OEM screen font is in use. The screen font can be changed with the Screen Font command.

To edit a value, position the text cursor over the two-digit hex value that is to be changed and type a new hex value. Alternatively, the cursor can be placed in the ASCII area at the right and characters can be keyed directly from the keyboard. The Tab key can be used to jump between equivalent positions in the hex and ASCII display areas.

In most cases, *binary files* should not be shortened or lengthened. Doing so will often invalidate the format of the file. If you are editing a file whose length *can* be safely altered, and you wish to do so, the right-click hex mode context menu provides options for inserting and deleting one or more bytes into/from the file.

When editing in hex mode, the Find and Replace commands will each present a special dialog which is designed for use in hex mode. In hex mode, these commands will permit strings to be entered as either a sequence two-digit hex codes, or as conventional text strings.

☞ Boxer will automatically use the Open Hex command when asked to open files with the extension `.COM`, `.EXE` or `.DLL`. Files with these extensions are assumed to be *binary files*, and cannot be edited by Boxer in conventional text mode.

The View | Hex Ruler command displays a hex ruler across the top of the screen.

The Go to Byte Offset command is sensitive to Hex Mode, and can be used to jump quickly to a specified offset within the file.

See the Open command for details on using Boxer's custom file open dialog and the standard Windows file open dialog.

☞ The maximum size for a file opened in hex mode is smaller than the usual file size limit. The limit is approximately 478 MB. This limit derives from the fact that the display of 16 characters in hex mode requires approximately 80 characters of storage space.

5.181 Open Program at Cursor

Menu: File > Open Other > Program at Cursor

Default Shortcut Key: none

Macro function: OpenProgramAtCursor()

The Open Program at Cursor command can be used to open the program or document that appears at the text cursor. For example, if the filepath to a `.PDF` file appears at the text cursor, this command will open that file in Acrobat Reader. If a `.DOC` file appears at the cursor, the file will be opened in Microsoft Word.

☞ This command relies on the underlying ability of the operating system to identify and locate the program that is associated with a given file type. If a document does not have an associated program, it cannot be opened. See File Associations for more details.

5.182 Open System Files

Menu: File > Open Other > System Files

Default Shortcut Key: none

Macro function: OpenSystemFiles()

The Open System Files command automatically loads a variety of operating system files into the editor. On Windows 95, Windows 98 and Windows Me, the following files are opened:

```
autoexec.bat
config.sys
system.ini
win.ini
```

On Windows NT, 2000 and XP, the following files are also opened:

```
config.nt
autoexec.nt
```

If the command is invoked while one or more system files are already open, an option is provided to close these files.

It is advisable to exercise extreme caution when editing files of this nature, and to always keep a backup copy of the original file.

5.183 Open URL at Cursor

Menu: File > Open Other > URL at Cursor

Default Shortcut Key: Ctrl+U

Macro function: OpenURLAtCursor()

The Open URL at Cursor command will attempt to launch your Internet browser to view the *URL* address beneath the text cursor.

This command can also be invoked by double clicking with the mouse on a URL which appears within text. The mouse cursor will change to the pointing hand when atop a URL to indicate that the address has been recognized.

In order to launch your Internet browser, Boxer relies upon the operating system shell's ability to open an Internet address. When an Internet browser is installed, it typically establishes itself as the program which is called by the shell to open such addresses. This is true of all common browsers you are likely to encounter. If you find that your Internet browser is *not* launched by Boxer, or if some other inactive browser is launched instead, it's because your active browser has not established itself as the one that processes the 'open' request from the operating system shell for Internet addresses. This situation should be rare, cannot be remedied by Boxer, and is not due to any shortcomings in Boxer.

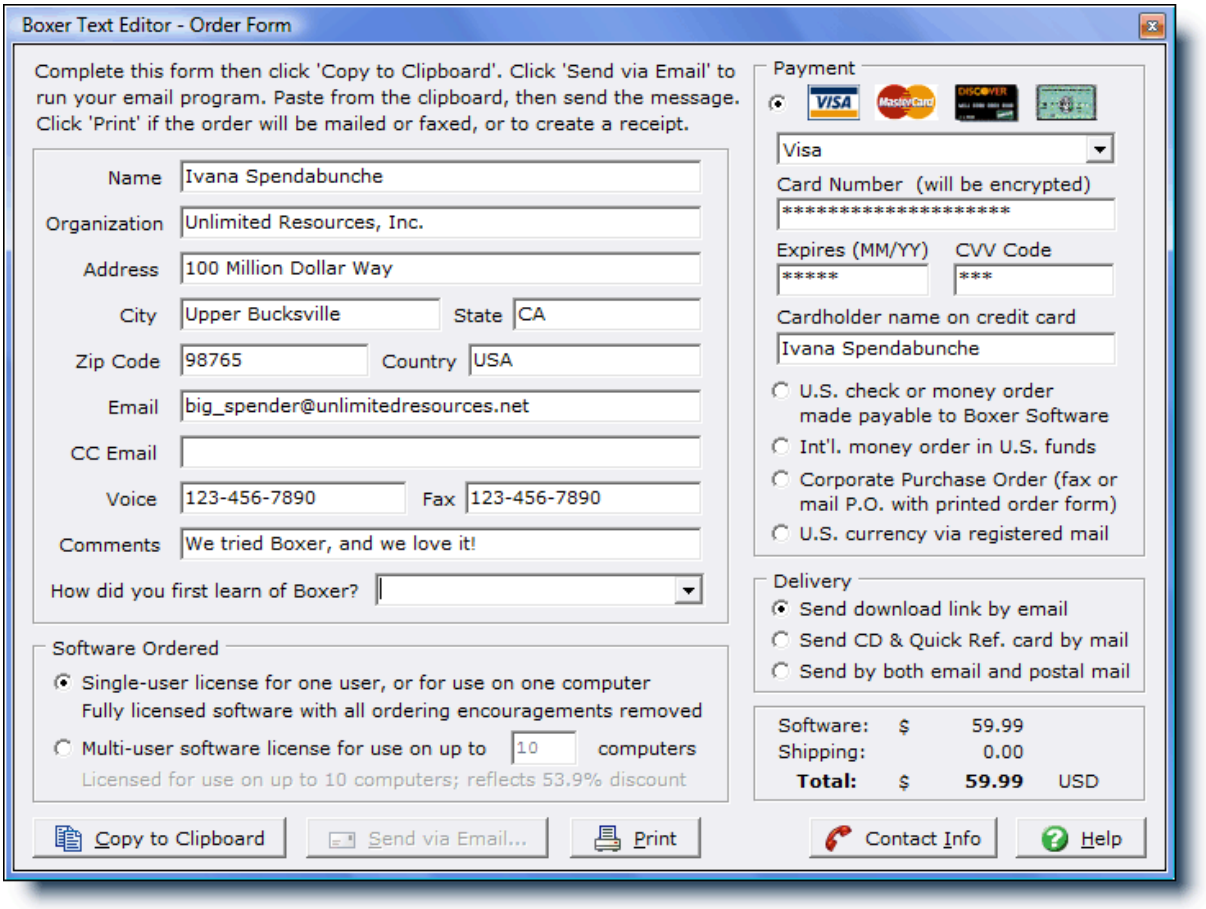
Boxer also supports opening local files with URLs of the form:
`file://c:\website\index.html`

5.184 Order Boxer

Menu: Help > Order Boxer

Default Shortcut Key: none

Boxer has an in-software order form to make ordering fast and easy. The order form is available by selecting the *Order Boxer* option from the Help menu, or by clicking the dollar bill icon on the toolbar of the evaluation version.



This order form can be used to submit your order by email, or to print an order form which can later be faxed or mailed along with payment. In all cases you can be assured that your order will receive prompt attention, and that we will safeguard your personal information. Boxer Software does not share its customers' mailing addresses, or email addresses, with any third parties.

If you prefer to print an order form from within this help file, and then mail or fax it to us, use this order form.

The Order Form will compute your total automatically as you complete your order. If a Multi-User License is being ordered, click the appropriate option and enter the quantity desired. The total will be updated automatically to reflect the quantity ordered. Likewise, shipping is computed according to the destination country, and depending on whether delivery will be made by email or postal mail (delivery by email is free). If you elect to have the software sent via email, an http link will be sent from which you can download the software; the software is not sent by email attachment.

Ordering by Email

Complete the form, and then click *Copy to Clipboard* to copy the information entered to the Windows clipboard. Click *Send via Email* to launch your email program. The *To* field of your email program should auto-fill with sales@boxersoftware.com. Paste the order information from the clipboard into the message body and send the message in the usual way. Note that your credit card information will be encoded using a proprietary encoding algorithm for added security. We will decode the information after your order arrives.

☞ If your email program does not launch after clicking *Send via Email*, simply start it in the usual way and paste the content on the clipboard into the body of a new message. Send the message to sales@boxersoftware.com.

Ordering at our Website

Visit www.boxersoftware.com to order from our secure order page. Full ordering details are provided at the site.

Ordering by Phone

Call toll-free within the U.S. and Canada at `1-800-98-BOXER (1-800-982-6937)` to order. Have your credit card ready; our sales representative will prompt you for the required information. From outside the U.S. and Canada call `+1-602-485-1635`. Business hours are Monday through Friday, 9 AM to 5 PM MST.

Ordering by Fax

Complete the form, and then click *Print*. Fax the order form to Boxer Software at `+1-602-485-1636`. Note that your credit card information will be encoded with a proprietary encoding algorithm for added security. We will decode the information after your order arrives. Our fax line is available 24 hours a day.

Ordering by Mail

Complete the form, and then click *Print*. Mail the order form to `Boxer Software, PO Box 14545, Scottsdale, AZ 85267-4545`. Note that your credit card information will be encoded with a proprietary encoding algorithm for added security. We will decode the information when your order arrives.

Ordering from Overseas

International Agents are available for those who might prefer to place their order with a local agent. Our agents accept payment in local currency and ship product from stock. Technical support services are also available.

Payment

Payment can be made in a variety of ways:

Credit Card
Visa, MasterCard, Discover or American Express

U.S. Check or Money Order
Made payable to 'Boxer Software'

Purchase Order
Purchase Orders can be mailed or faxed. Please make sure the Purchase Order includes both the shipping and invoicing addresses. Our payment terms are Net 30 days.

Western Union
Wire funds to 'David Hamel' and tell us the Control Number for the transaction, as well as the sender's name and the exact amount sent. Western Union also allows wire transfers to be made from their website: www.westernunion.com

U.S. Cash
Sent by certified or registered mail

PayPal
Send funds to 'sales@boxersoftware.com'. Don't have PayPal yet? Click here to sign up: www.paypal.com

International Money Order
Available at most banks. Money order should be drawn on a U.S. bank, in U.S. Funds, payable to 'Boxer Software'

International Postal Money Order
Available at the Post Office. Money order should be drawn in U.S. Funds, payable to 'Boxer Software'

Bank Wire Transfer
Please contact us for current bank transfer information. A $5.00 surcharge must be added to help offset the wire transfer fees we are assessed by our bank.
(**Note:** U.S. banks are not nearly as efficient as European banks with regard to bank wire transfers. Incoming transfers are slow, and receiving fees can be as high as $20.00. For this reason, we strongly encourage using another method of payment.)

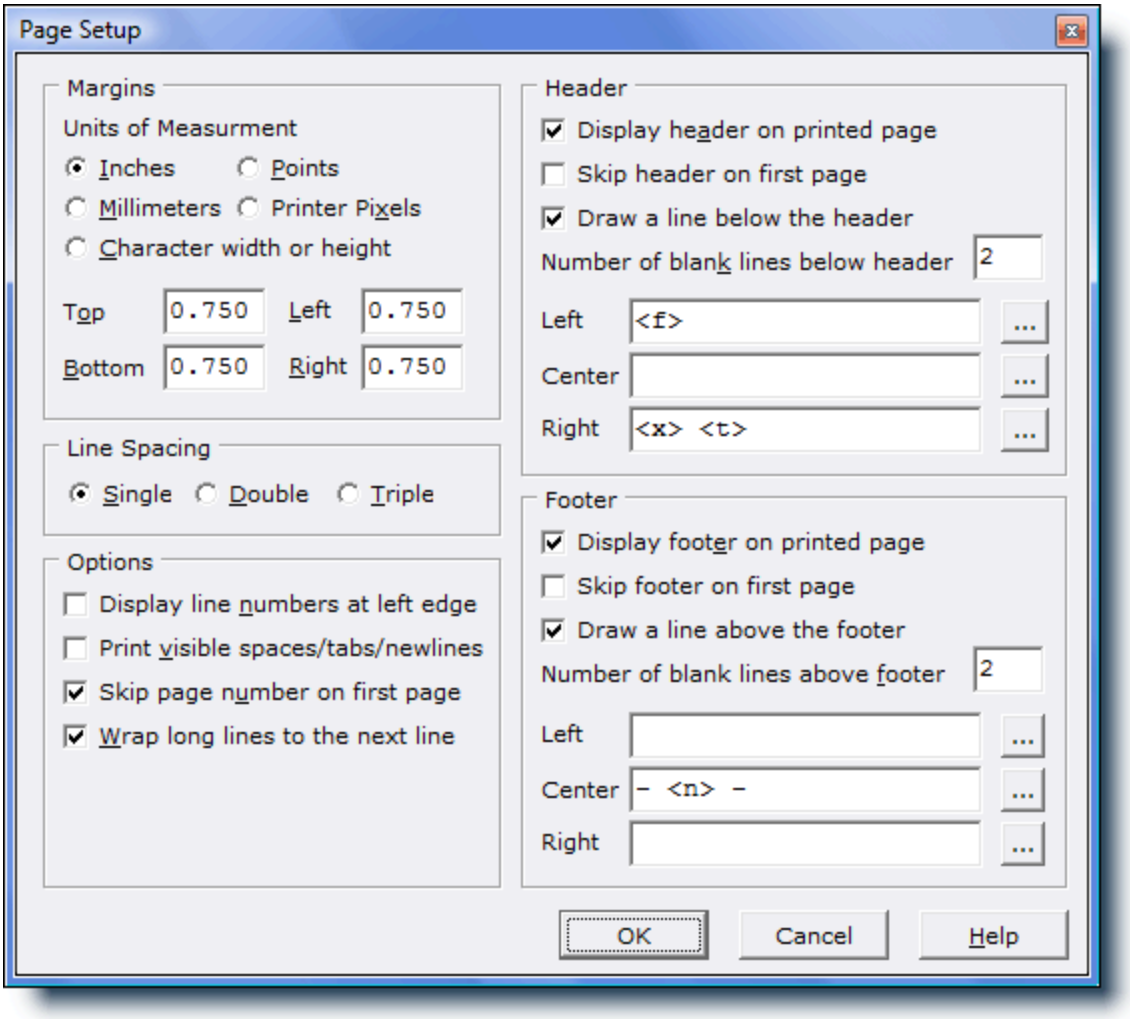
5.185 Page Setup

Menu: File > Page Setup

Default Shortcut Key: none

Macro function: PageSetup()

The Page Setup command provides access to a dialog box which controls the layout of the printed page. The following controls are provided:



The settings made using the Page Setup command will be used for all print jobs that are performed from within Boxer. In other words, the Page Setup settings belong to the editor, and not to the current file. This method of operation differs from that of a Page Setup command within a word processor. A word processor is able to save its page settings within each document because the format of its documents is proprietary. As a text editor Boxer must save its files in ASCII format, and therefore cannot embed such information in the files it creates.

Margins

Inches
The margin values will be entered in inches. Use floating point values if needed (2.50), but not fractional values (2-1/2).

Millimeters
The margin values will be entered in millimeters.

Points
The margin values will be entered in points. There are 72 points to the inch.

Printer Pixels
The margin values will be entered in printer pixels. The number of pixels (dots) per inch will depend on the printer. Most printers are either 300 dpi or 600 dpi. This option provides the finest control over margin sizes.

Character width or height
The margin values are related to the width or height of a printed character. The Top and Bottom margin values will dictate the number of lines of margin. The Left and Right margin values will dictate the number of character widths of margin.

☞ Margin values can be no smaller than the printer's *non-printable zone*, which is the area at each edge of the paper on which the printer is physically incapable of printing. Boxer computes this value automatically, so you do not need to account for it when specifying margin values.

Top
Use this option to specify the distance from the top edge of the printed page to the *header* line, or to top of the body text when a header line is not used.

Bottom
Use this option to specify the distance from the bottom edge of the printed page to the *footer* line, or to the bottom of the body text when a footer line is not used.

Left
Use this option to specify the distance from the left edge of the paper to the beginning of the body text, or line numbers (if applicable).

Right
Use this option to specify the distance from the right edge of the paper to the right edge of the body text. Depending on the state of the *Wrap long lines to next line* checkbox (see below), long lines will either be wrapped to the next line or truncated.

☞ Changing the right margin value does not influence the wrap column of preformatted text as might occur within a Word Processor. Because Boxer is a Text Editor, not a Word Processor, the user alone controls hard line enders, and these are never re-wrapped without your knowledge. It may be necessary to experiment with different Printer Font sizes and/or different Text Width values to achieve optimum results on the printed page. The Print Preview command will be useful in this regard, as it enables you to preview a print job on-screen without sending it to the printer.

Line Spacing

Select from **Single**, **Double** or **Triple**. Double and triple might be used when submitting a document which will be marked up by a teacher, for example.

Options

Display line numbers at left edge
If checked, line numbers will be applied at the left edge of the printed page. Overflow

lines will be marked with a '+', and will therefore not alter the actual line count unnaturally.

Print visible spaces/tabs/newlines
If checked, Spaces, Tabs and Newline characters will be appear on the printed page as visible characters, using the same characters as have been configured for the Visible Spaces command.

The symbols which are used to represent Spaces, Tabs and Newlines are user-configurable. These can be set using options on the Configure | Preferences | Display options page. Separate options are provided for use with both ANSI and OEM printer fonts.

Skip page number on first page
If this option is selected, the page number will not be printed on the first page.

Wrap long lines to the next line
If checked, long lines will be wrapped to the next line rather than being truncated. Using a smaller Printer Font is one way to cure overflow lines. Another is to Reformat the document using a narrower Text Width.

Header

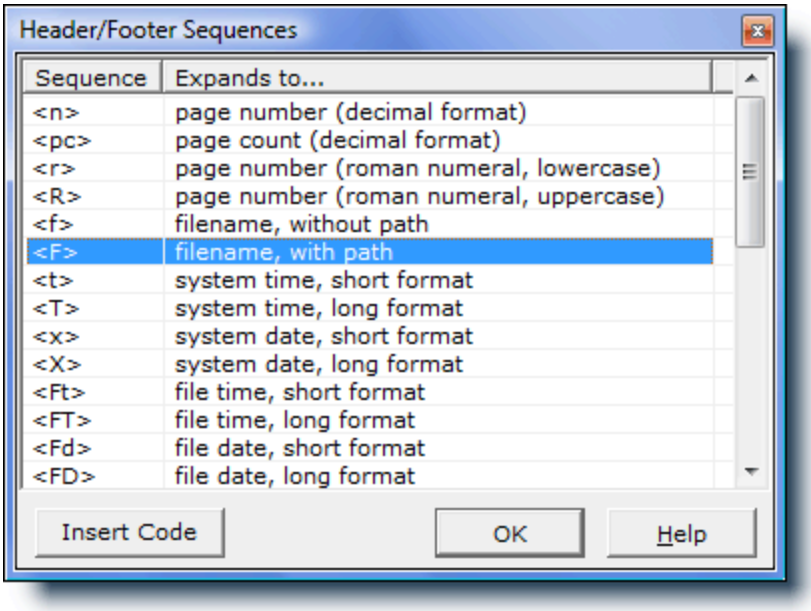
Display header on printed page
When checked, the related header controls become active and a *header* line will appear on all printed pages.

Skip header on first page
If checked, the header will not be printed on the first page.

Draw a line below the header
Use this option to cause a thin solid line to appear just the below the line of header text.

Number of blank lines below header
This option controls the number of lines of spacing between the header line and the top of the body text.

Left / Center / Right
These fields allow the text that is to appear in each header zone to be specified. You may enter any text you like or use one or more of several pre-defined substitution sequences to insert a page number, filename, time, date, etc. Clicking the button to the right of each field displays the available sequences:



Footer

Display footer on printed page
When checked, the related footer controls become active and a *footer* line will appear on all printed pages.

Skip footer on first page
If checked, the footer will not be printed on the first page.

Draw a line above the footer
Use this option to cause a thin solid line to appear just above the line of footer text.

Number of blank lines above footer
This option controls the number of lines of spacing between the footer line and the bottom of the body text.

In the evaluation version of Boxer, a 'watermark' will appear on all printed pages between the footer line and the bottom of the body text. This line serves as both a reminder and an encouragement to order a fully licensed copy. This reminder line is of course not present in the fully licensed version of Boxer.

Left / Center / Right
These fields allow the text which is to appear in each footer zone to be specified. You may enter any text you like or use one or more of several pre-defined substitution sequences to insert a page number, filename, time, date, etc. Click the button to the right of each field to select from the list of available sequences.

5.186 Paste

Menu: Edit > Paste

Default Shortcut Key: Ctrl+V

Macro function: Paste()

The Paste command inserts the text from the current clipboard at the location of the text cursor in the current file. The current clipboard might be the Windows clipboard or one of Boxer's eight internal clipboards. See the Edit | Set Clipboard command for details on changing the current clipboard.

If stream text (Edit | Select Stream) is being pasted, the clipboard text is inserted as if it had been typed from the keyboard. That is, the character at the text cursor is pushed along as far as is needed to accommodate the new text.

If columnar text (Edit | Select Columnar) is being pasted, the method of insertion is sensitive to the current edit mode. In Insert mode text will be pushed right to accommodate the size of the rectangular block being inserted. In Typeover mode the clipboard text will overwrite any text which may exist in the destination rectangle.

If an entire line has been placed on the clipboard by using the Copy or Cut command *without* first selecting text, the text cursor will be positioned to the start of line before the text is inserted.

The placement of the text cursor after a Paste operation can be controlled with an option on the Configure | Preferences | Editing 1 options page. The option is titled *Stay at insertion point when Pasting*.


If the Paste command is issued when no files are open, and when text is present on the active clipboard, a new file will be created automatically and the clipboard text will be pasted into that file.


When the Paste command is issued repeatedly to paste the same clipboard content, the status line will report a count of the number of times that the content has been pasted.


When placing columnar text onto a clipboard, Boxer must take care so that subsequent Paste operations of that text will be performed properly. Columnar clipboard text must be pasted differently than stream text, since all lines must move rightward by the width of the text block. Notwithstanding this fact, columnar clipboard text placed onto the Windows clipboard by Boxer can still be pasted into other Windows applications. Boxer does not use a *private clipboard format* for this purpose.

5.187 Paste As

Menu: Edit > Paste As

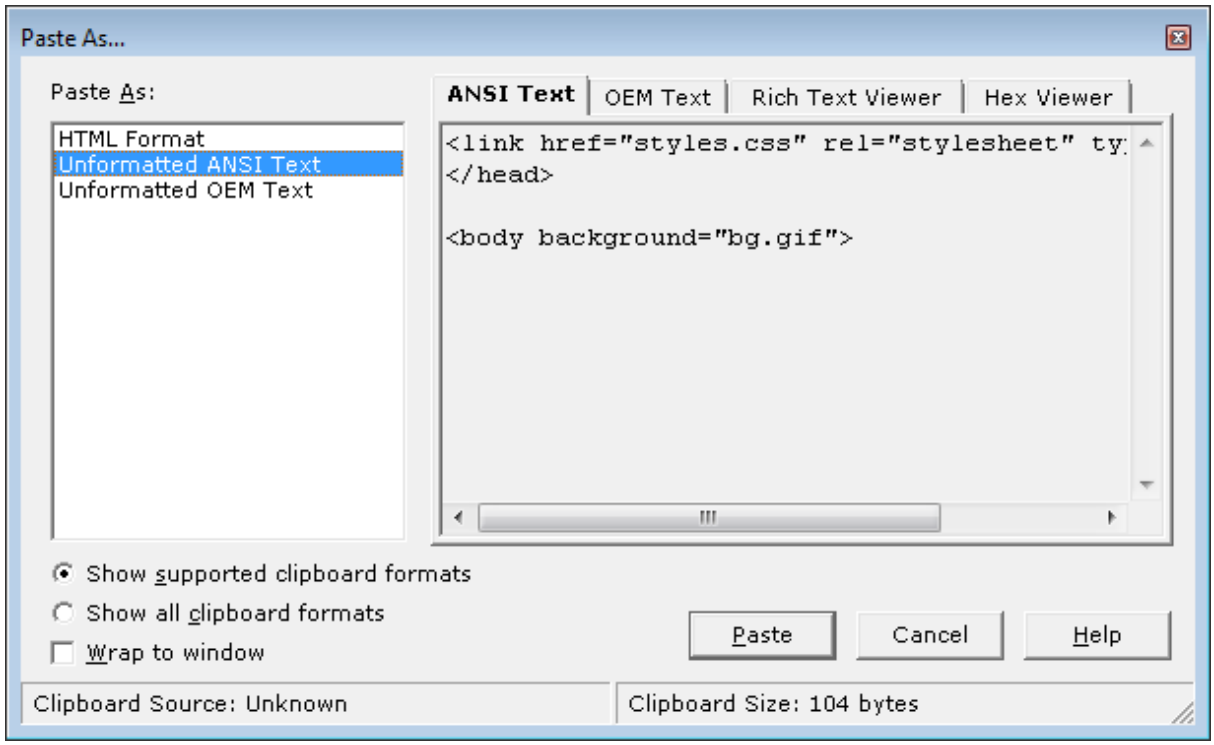
Default Shortcut Key: Shift+Ctrl+V

Macro function: PasteAs()

The Paste As command allows the content of the Windows clipboard to be viewed in

various formats, and with various viewers. This makes it possible to Paste text into Boxer in formats other than normal text. After selecting the desired format, simply click the *Paste* button.

Background: the Windows clipboard is capable of holding multiple pieces of data simultaneously. Sometimes this capability is exploited so that different versions of the same data can be made available. In other cases, different formats will hold more descriptive copies of the same data. For example, when you copy text from a web page that is being displayed in Internet Explorer, the data will be placed on the clipboard in several different formats. These formats are HTML Format, Rich Text Format (RTF), ANSI Text and OEM Text. When another application pastes that data from the clipboard, it chooses the format that is most meaningful to that application. Boxer's Paste command would use the ANSI Text format. This is where the Paste As command becomes useful. There might be times when you would prefer that Boxer be able to paste the clipboard data in HTML format, so that HTML formatting codes and hyperlinks are not lost. For some users, having access to the data in RTF format might prove useful. The Paste As command allows all available formats to be viewed so that the most useful format can be used.



Boxer's clipboard commands will sense the type of text data being placed on the Windows clipboard and, when appropriate, use a more descriptive clipboard format to tag that data. For example, when Boxer senses that HTML code is being copied to the clipboard, the text will also be placed in the HTML clipboard format. This enables a conforming program to paste the data more intelligently. Boxer will also tag Rich Text data (RTF) and Comma-Separated Value (CSV).

5.188 Paste Clipboard

Menu: Edit > Paste Clipboard > Clipboard *n*

Default Shortcut Key: Shift+Alt+*n*

Macro function: PasteClipboard()

The Paste Clipboard commands permit the content of any clipboard to be inserted into the text file directly, without the need to first select that clipboard with the Set Clipboard command before using Paste.

The content of Boxer's internal clipboards is saved from session to session (subject to a limit; see Sizes and Limits), ensuring that their content is always available. Since the Paste Clipboard commands allow pasting from any clipboard with a single key sequence, the internal clipboards can be the ideal place to store commonly used text blocks.

When the content of a clipboard is displayed in a popup window, the text is displayed with an 8 point, fixed width, *Courier New* font. This font utilizes the ANSI character set mapping. If the current screen font uses an OEM character set mapping, and if characters outside the normal alphanumeric range reside on the clipboard, then the content of the clipboard may appear different in the popup window than it would in the underlying file. This difference is simply the result of a difference in character sets, and does not mean that the data on the clipboard has been adjusted or corrupted.

5.189 Pause Recording

Menu: Tools > Pause Recording

Default Shortcut Key: none

Macro function: none

Use the Pause Recording command to temporarily stop a keystroke recording that's already in process. Once paused, editor commands are not stored in the active recording. To resume recording, simply issue this command again: its name in the main menu changes to *Resume Recording* when recording is paused.

5.190 Playback Keys

Menu: Tools > Playback Keys

Default Shortcut Key: F5

Macro function: none

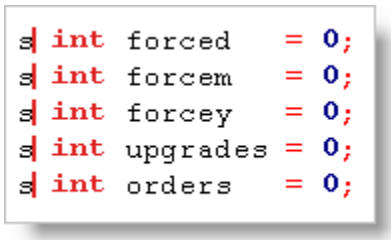
Issue the Playback Keys command to playback the most recent key recording. Execution begins immediately, so make sure that the text cursor is positioned as desired before proceeding.

5.191 Power Columns

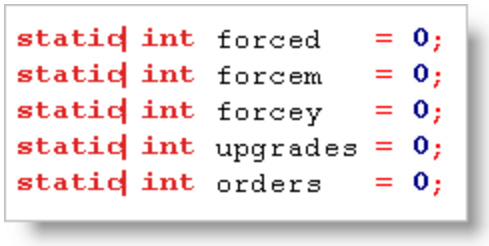
Boxer's Power Columns feature can be a big time saver when editing text that requires identical changes to be made on each line. In Power Columns mode, the text you type is applied to every line within the range of selected lines. When you cursor left or right, the insertion point moves in all selected lines. If you press *Delete*, or *Backspace*, a character is deleted in each line. You can even Paste a short text string into each line with just a single Paste command. Power Columns can work on two lines, two thousand lines... or even more.

Let's consider an example... Suppose you've got five variable declarations that need to have the word 'static' applied to each of them. This is an editing task that arises in programming, but you'll probably be able to imagine other uses as well. To enter Power Columns mode, you simply need to create a Columnar Selection of zero width. Make sure the selection mode is set to Columnar on the Block menu, and then press *Shift+Down* four times. This will be the result:

The special red cursor bars indicate that Power Columns mode is active. When you press the letter 's', the character is inserted on each line:



When you type 'tatic' the rest of the text is entered, and the job is done:



(The word 'static' changed to red because 'static' is a reserved word.)

If you make a mistake while typing, the *Delete* and *Backspace* keys operate predictably to correct your error. If additional changes are needed in another portion of the line, the *Left* and *Right* arrow keys can be used to move the insertion point while still remaining in Power Columns mode. To exit Power Columns mode, press *Escape*, or use the *Up* or *Down* arrow.

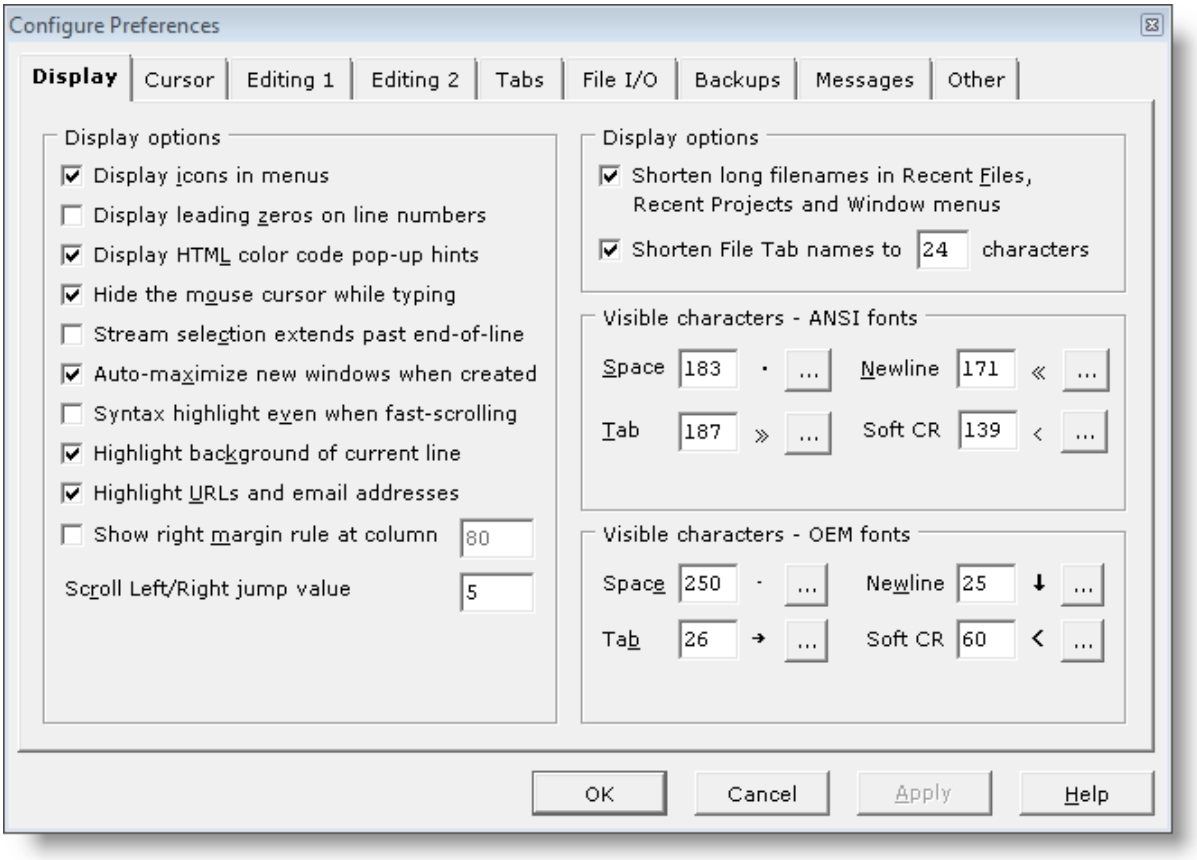
5.192 Preferences - Display

Menu: Configure > Preferences

Default Shortcut Key: none

Macro function: ConfigurePreferences()

The Display page of the Configure Preferences dialog box contains options related to the appearance of Boxer's screen and the display of edited files:



Display Options

Display icons in menus
Use this option to control the display of icons within the main menus.

Display leading zeros on line numbers
This option controls whether or not leading zeros will be used when the View | Line Numbers command is in use.

Display HTML Color Code pop-up hints
This option controls whether or not Boxer will display a pop-up HTML Color Code Hint when the mouse cursor hovers atop an HTML Color Code (such as `color="#FF3B80"` or `color="DarkSlateBlue"`). The pop-up box shows the color associated with the color sequence below the mouse cursor.

Hide the mouse cursor while typing
Use this option to control whether or not the mouse cursor will be hidden while text is being entered from the keyboard. If hidden, the mouse cursor will be redisplayed when it is moved.

Stream selection extends past end-of-line
This option controls the way in which a multi-line Stream selection is displayed on-screen. When this option is active, selected lines will be highlighted all the way to

the right edge of the window, even when lines are shorter than the window's width. When this option is inactive, selected lines will be highlighted only up to the end of each line. This option has no effect on the text included within a selection. It is simply a display option.

Auto-maximize new windows when created
Use this option to cause new windows created with File | New or File | Open to be opened in maximized form.

Syntax highlight even when fast-scrolling
This option controls whether or not Boxer will perform Syntax Highlighting while a file is scrolling rapidly, due to a keyboard key being held down. Scrolling will be faster if Boxer is allowed to suspend Syntax Highlighting in this situation. The screen will be updated instantly when the scrolling key is released.

Highlight background of current line
Use this option to control whether or not the background of the current will be displayed in a different color. Doing so can make it easier to locate the current line. The Configure Colors command can be used to select the background color used.

Highlight URLs and email addresses
This option controls whether or not Boxer will apply coloration to URLs and email addresses when they are encountered within ordinary text files. The color and font style used to highlight an address can be controlled with the Configure | Colors command. Disabling highlighting also disables the ability to double-click on these addresses in order to launch an internet browser or email client. In such case, the Open File in Browser and Open Email at Cursor commands could be used instead.

Show right margin rule at column...
Use this option to control the display of the Right Margin Rule, and to set the column at which the rule is displayed.

Scroll Left/Right jump value
This option controls the number of columns that the Scroll Left and Scroll Right commands will jump by when panning the screen left of right..

Shorten long filenames in File/Project/Window menus
Use this option to control whether or not long filenames will be shortened when they appear in the either the Recent Files, Recent Projects or Window menus. If this option is active, long file names will be shortened whenever they exceed 60 characters in length.

Shorten File Tab names to *n* characters
Use this option to control whether or not the filenames displayed in Boxer's File Tabs will be shortened when they exceed a specified length. When filename shortening is required, as many as four characters will be retained from the file extension, with the balance of characters being retained from the left side of the filename. Missing characters in the middle of the filename will be replaced by three dots (. . .).

Visible characters - ANSI fonts

Space value
Tab value
Newline value
These options can be used to designate the characters which will be used for Visible Spaces display when an ANSI Screen Font is in use. Use the button with the ellipsis (...) to select a character from the ANSI Chart.

Visible characters - OEM fonts

Space value
Tab value
Newline value
These options can be used to designate the characters which will be used for Visible Spaces display when an OEM Screen Font is in use. Use the button with the ellipsis (...) to select a character from the OEM Chart.

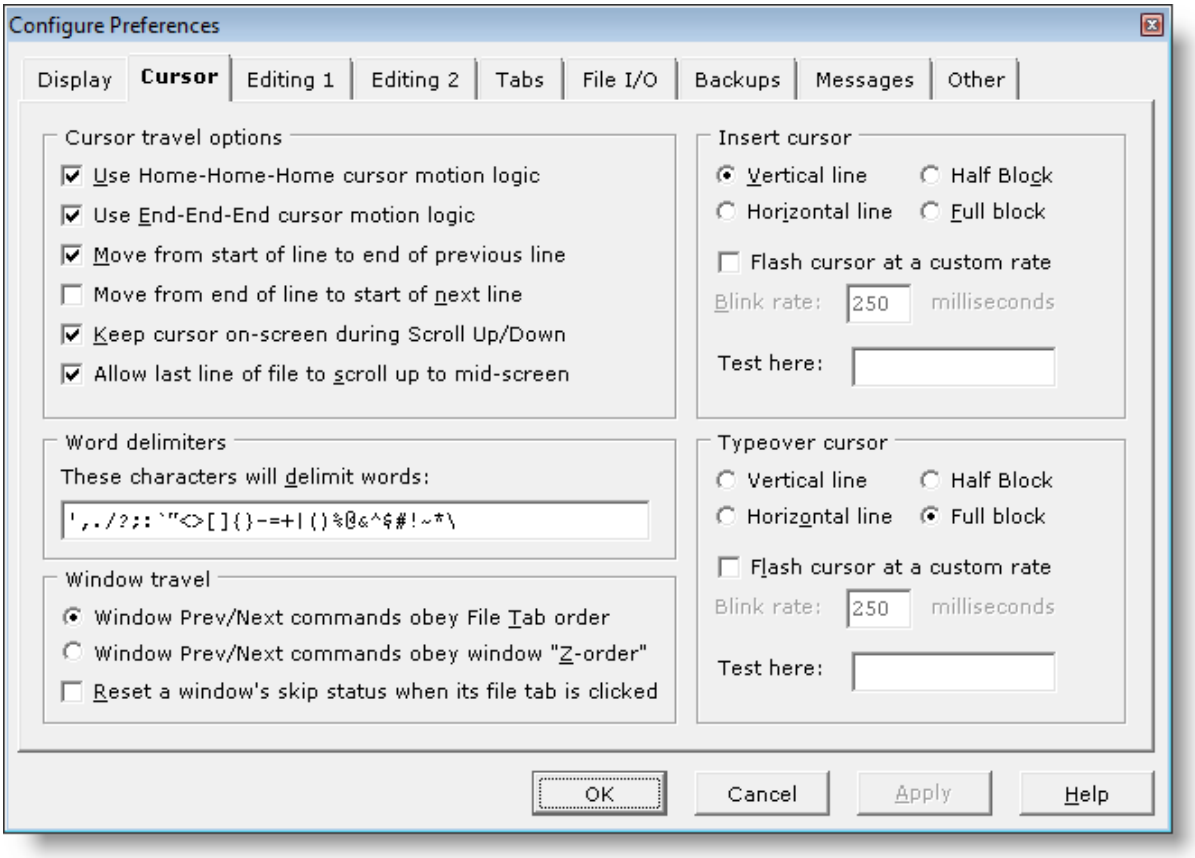
5.193 Preferences - Cursor

Menu: Configure > Preferences

Default Shortcut Key: none

Macro function: ConfigurePreferences()

The Cursor page of the Configure Preferences dialog box contains options which can be used to control cursor movement within the file:



Cursor Travel Options

Use Home-Home-Home cursor motion logic
When this option is active the function of the *Home* key becomes dependent on the number of times *Home* has been pressed. When *Home* is first pressed, the text cursor is moved to the beginning of the current line. When *Home* is pressed again, the cursor is moved to the first line in the window. When *Home* is pressed a third time, the cursor is moved to the first line in the file.

☞ If the *Home* key is not assigned to *Start of Line*, *Home-Home-Home* functionality becomes unavailable. Its function is tied directly to the *Home* key, and not to other keys which might be assigned to *Start of Line*.

Use End-End-End cursor motion logic
When this option is active the function of the *End* key becomes dependent on the number of times *End* has been pressed. When *End* is first pressed, the text cursor is moved to the end of the current line. When *End* is pressed again, the cursor is moved to the last line in the window. When *End* is pressed a third time, the cursor is moved to the last line in the file.

☞ If the *End* key is not assigned to *End of Line*, *End-End-End* functionality becomes unavailable. Its function is tied directly to the *End* key, and not to other keys which might be assigned to *End of Line*.

Move from start of line to end of previous line
Use this option to permit the text cursor to move from the start of the current line to the end of the previous line when the *Left Arrow* is pressed. When this option is inactive, pressing the *Left Arrow* in column one will result in no cursor movement.

Move from end of line to start of next line
Use this option to force the text cursor to move from the end of the current line to the start of the next line when the *Right Arrow* is pressed. When this option is inactive, the cursor is allowed to travel rightward past the end of a line.

This option also affects the way Boxer behaves when the *Up Arrow* and *Down Arrow* are used to cursor across lines of varying lengths. If this option is active, the column of the text cursor will be adjusted when moving onto a line that is shorter than the current column. If this option is inactive, the text cursor column will be maintained when moving from line to line.

Keep cursor on-screen during Scroll Up/Down
Use this option to control how the position of the text cursor is treated when using the Scroll Up and/or Scroll Down commands. When this option is checked, the text cursor will be moved (if necessary) to keep it on-screen while scrolling. When this option is not checked the cursor position will be maintained even when the line containing the text cursor is scrolled outside the view of the window.

Allow last line of file to scroll up to mid-screen
When this option is on, using the Down arrow to scroll to end of file will cause the last line of the file to appear as high as mid-screen. When this option is off, the last line of the file will not scroll up past the bottom of the window.

Word delimiters

These characters will delimit words
This option can be used to designate the characters which are considered to be *word delimiters*. Word delimiters are those characters which serve to separate one word from another. It may be desirable to add or remove symbols from the default delimiter list in order to improve the behavior of the *Word Left* and *Word Right* commands within certain types of file.

The word delimiter list is used by various commands to determine the extent of their operation. Among these commands are Word Left, Word Right, Delete Previous Word, Delete Next Word, Swap Words, Open Filename at Cursor, Open URL at Cursor and Open Email at Cursor.

Window Travel

Window Previous/Next command obey File Tab order
Window Previous/Next command obey window 'Z-order'
Use these options to control how Boxer responds to the Window Previous and Window Next commands. When *File Tab order* is selected, the window commands will use the ordering of the File Tabs to determine which window to move to. When the *Z-order* option is selected, window movement will be determined according to an order

maintained by Windows. A window is promoted in the Z-order when it is made current. Less frequently used windows will gradually fall to the bottom of the z-order.

Reset a window's skip status when its file tab is clicked
When this option is on, clicking on a file tab will cause its skip status to be reset to normal.

Insert cursor and Typeover cursor

The shape and flash rate can be set independently for the Insert and Typeover cursors.

Vertical Line
Use this option to set the text cursor to a thin vertical line which sits at the left edge of the character cell.

Horizontal Line
Use this option to set the text cursor to a horizontal line which sits at the base of the character cell.

Half Block
Use this option to set the text cursor to a block which occupies the lower half of the character cell.

Full Block
Use this option to set the text cursor to a block which occupies the full character cell.

Flash cursor at a custom rate
Use this option to set the rate at which the cursor flashes. A *millisecond* is one thousandth of a second.

Test here
Use this edit box to test the new shape and flash rate.


Changes made to the text cursor apply only to Boxer, and do not affect other applications.

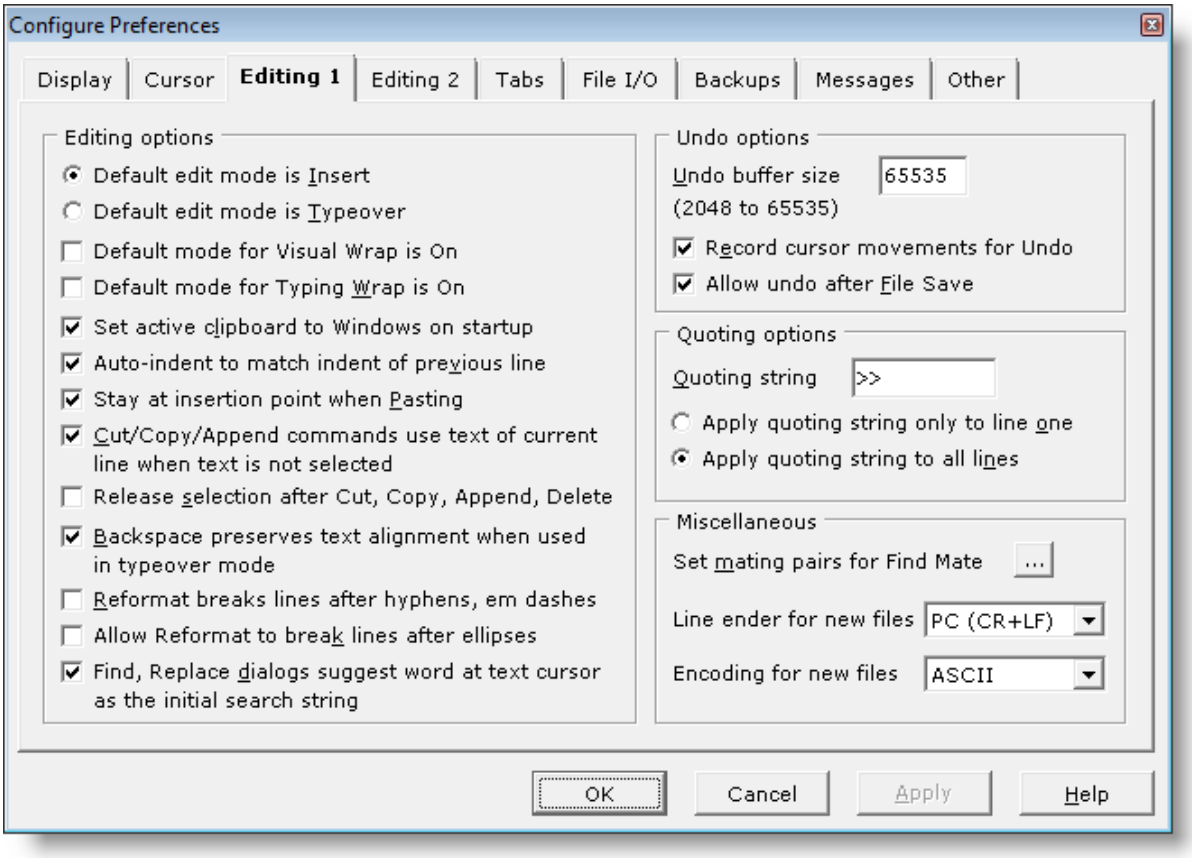
5.194 Preferences - Editing 1

Menu: Configure > Preferences

Default Shortcut Key: none

Macro function: ConfigurePreferences()

The Editing 1 page of the Configure Preferences dialog box contains options which relate to the editing of text files:



Editing options

Default edit mode is Insert
Use this option to set the default edit mode to *Insert*. In Insert mode, existing text is pushed rightward to make room for characters which are typed from the keyboard. Note that this option causes newly created windows to begin in Insert mode, but it does not change the edit mode of any editor windows that may already be open.

Default edit mode is Typeover
Use this option to set the default edit mode to *Typeover*. In Typeover mode, characters which are typed from the keyboard replace existing text. Note that this option causes newly created windows to begin in Typeover mode, but it does not change the edit mode of any editor windows that may already be open.

Default mode for Visual Wrap mode is On
Use this option to change the default Visual Wrap setting to on. Note that this option causes newly created windows to begin with Visual Wrap on, but it does not change the Visual Wrap mode of any editor windows that may already be open. The Visual Wrap command can always be used to change the setting for the current editor window, independent of this option.

Default mode for Typing Wrap mode is On
Use this option to change the default Typing Wrap setting to on. Note that this option causes newly created windows to begin with Typing Wrap on, but it does not change the

Typing Wrap mode of any editor windows that may already be open. The Typing Wrap command can always be used to change the setting for the current editor window, independent of this option.

Set active clipboard to Windows on startup
When checked, this options ensures that the active clipboard will be restored to the Windows clipboard each time Boxer is started.

Auto-indent to match indent of previous line
Use this option to enable *Auto-indent*. When Auto-Indent is active, pressing *Enter* at the end of a line will place the text cursor on a new line below with an indent level equal to that of the line above.

Stay at insertion point when Pasting
Use this option to cause the text cursor to remain at the point of insertion following a Paste operation. If inactive, the text cursor is placed at the end of the text which was pasted.

Cut/Copy/Append commands use text of current line when text is not selected
This option permits the Cut, Copy, Append and Cut Append commands to operate on the current line as though it were selected. Simply issue the desired command from any point on the line and the operation will be performed as though the whole line were selected.

Release selection after Cut, Copy, Append, Delete
When selected, this options causes a text selection to be automatically released after a clipboard operation is performed.

Backspace preserves text alignment when used in Typeover mode
Use this option to make the Backspace key overwrite with spaces when used in Typeover mode.

Allow Reformat to break lines after hyphens and em-dashes
When selected, this option allows the Paragraph | Reformat command to break a line after a hyphen (-) or an em-dash (--).

Allow Reformat to break lines after ellipses
When selected, this option allows the Paragraph | Reformat command to break a line after an ellipsis (...).

Find, Replace dialogs suggest word at text cursor as the initial search string
When selected, this option controls whether the Find, Replace, Find Text in Disk Files and Replace Line Enders dialogs will insert the word at the text cursor into the find edit box.

Undo Options

Undo buffer size
Use this command to set the buffer size used by the Undo command. Values between 2048 and 65535 are permitted. This value represents the amount of memory (in bytes) which is reserved for tracking undo information.

The default value is 65535, which is also the maximum value. There is little reason to select smaller values, as the memory cost is small compared to the utility that Undo provides.

Allow undo after File Save
Use this option to indicate that the Undo command should remain operable after the Save command is used, thereby allowing changes which occurred before Save to be undone. If this option is inactive, the Save command has the effect of the Clear Undo command, since Undo information is lost for changes made before the save.

Quoting Options

Quoting String
Use this option to specify the symbol (or symbols) which are to be used by the Quote and Reformat command during its operation.

☞ The Quote and Reformat command makes use of the Reformat command internally during its operation. As is noted in the Reformat command, lines beginning with a period (.) are treated as blank lines in order to recognize text markup tags. As a result, the use of a quoting string that begins with a period will not produce the desired results, and should be avoided. All other symbols and characters are permissible.

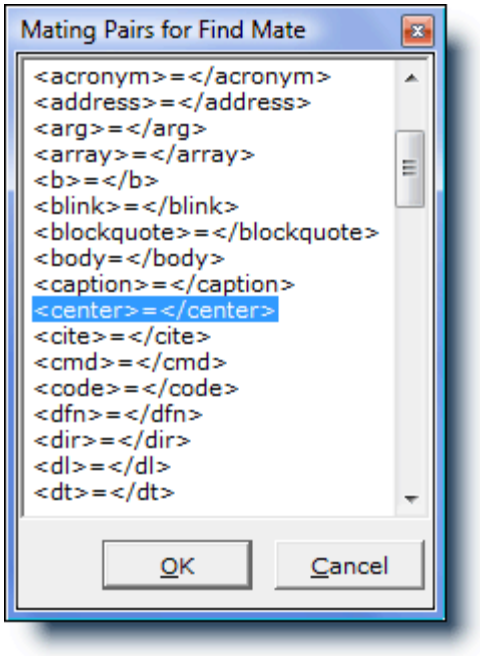
Apply quoting string only to line one
Use this option to designate that the Quote and Reformat command apply the *Quoting String* to the first line of the paragraph quoted.

Apply quoting string to all lines
Use this option to designate that the Quote and Reformat command apply the *Quoting String* to all lines within the paragraph quoted.

Miscellaneous

Set mating pairs for Find Mate
This option can be used to edit the mating pairs which are used by the Find Mate command. The Find Mate command is used to jump quickly from a parenthetical element at the text cursor to its mate.

When the ellipsis (...) button is clicked, a small edit window appears which contains the currently defined pairs:



Each mating pair resides on a single line, with the equal sign (=) being used to separate the opening string from the closing string. Pairs can be removed from the list, or new pairs can be added. Click *OK* to save the changes.

Line ender for new files
This option can be used to set the default line ender type for newly created files. Choose from PC, Macintosh or Unix style line enders. A file's line ender can also be changed from the File | Properties dialog.

Encoding for new files
This option can be used to set the default file encoding format for newly created files. Choose from ASCII, UTF-8, UTF-16 little endian or UTF-16 big endian. A file's encoding format can also be changed from the File | Properties dialog.

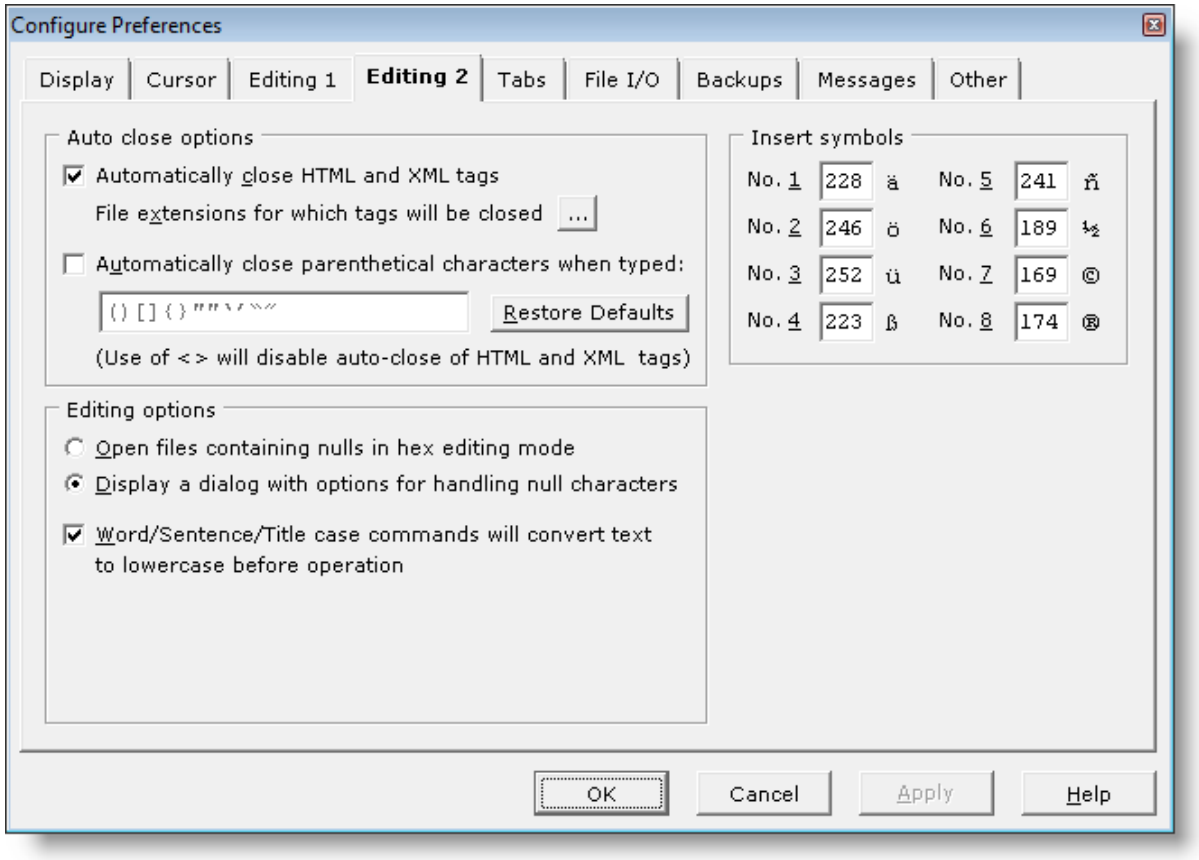
5.195 Preferences - Editing 2

Menu: Configure > Preferences

Default Shortcut Key: none

Macro function: ConfigurePreferences()

The Editing 2 page of the Configure Preferences dialog box contains options which relate to the editing of text files:



Auto close options

Auto Close HTML, XML tags
This option can be used to enable or disable Boxer's Auto Tag Close feature. When enabled, and when an eligible file type is being edited, Boxer will automatically create the closing tag and place the cursor between the tags. For example, when you type `<center>`, Boxer automatically completes the tag with `</center>` and places the text cursor between the tags.

Auto Tag Close file extensions

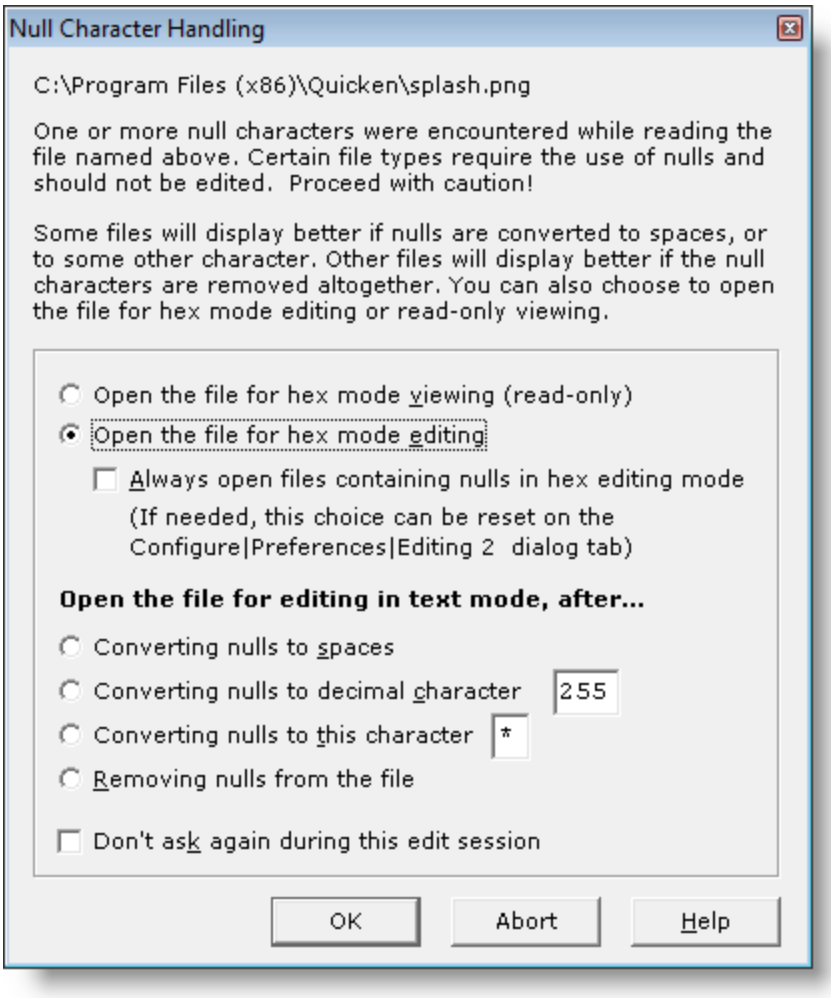
Use this option to control which file types are eligible for the Auto Tag Close feature. When the ellipsis (...) button is clicked, a small edit window appears which contains the eligible file extensions.

Automatically close parenthetical characters when typed
When this option is active, parenthetical characters will be automatically closed when they are typed from the keyboard. The text cursor is then placed between the mated characters. An edit box is provided to control which characters will be automatically closed. This option is off by default.

By necessity, if < and > are designated among the list of mating characters, the auto-close feature for HTML and XML tags will be disabled.

Editing options

Open files containing nulls in hex editing mode
Display a dialog with options for handling null characters
These options control how Boxer reacts when a request is made to open a file that contains null characters. If the first option is selected, Boxer will automatically open the file in hex editing mode. If the second option is selected, the Null Character Handling dialog will appear before the file is opened, providing additional options for how the file can be handled:



Word/Sentence/Title case commands will convert text to lowercase before operation

This option causes the Word, Sentence and Title case commands to automatically convert the selected text to lowercase before performing their function. When operating on uppercase text, this mode of operation allows the desired conversion to be performed in one step.

But take note: if the Word case command is applied to the following text:

```
IBM, eBay, MasterCard Report Record Profits
```

the result may not be as expected:

```
Ibm, Ebay, Mastercard Report Record Profits
```

Insert Symbols

The edit boxes within this section permit the definition of up to eight character values for use with the Insert Symbols feature. The values for the characters are entered in decimal format, and must reside in the range 1 to 255. The defined character is displayed to the right of each edit box using the same character set (ANSI or OEM) that is in use in the editor itself. This should help ensure that the characters are displayed as expected. If a character does not display within the dialog with the expected representation, this should not be cause for alarm. Simply verify that the character has the expected appearance when inserted into the text file.

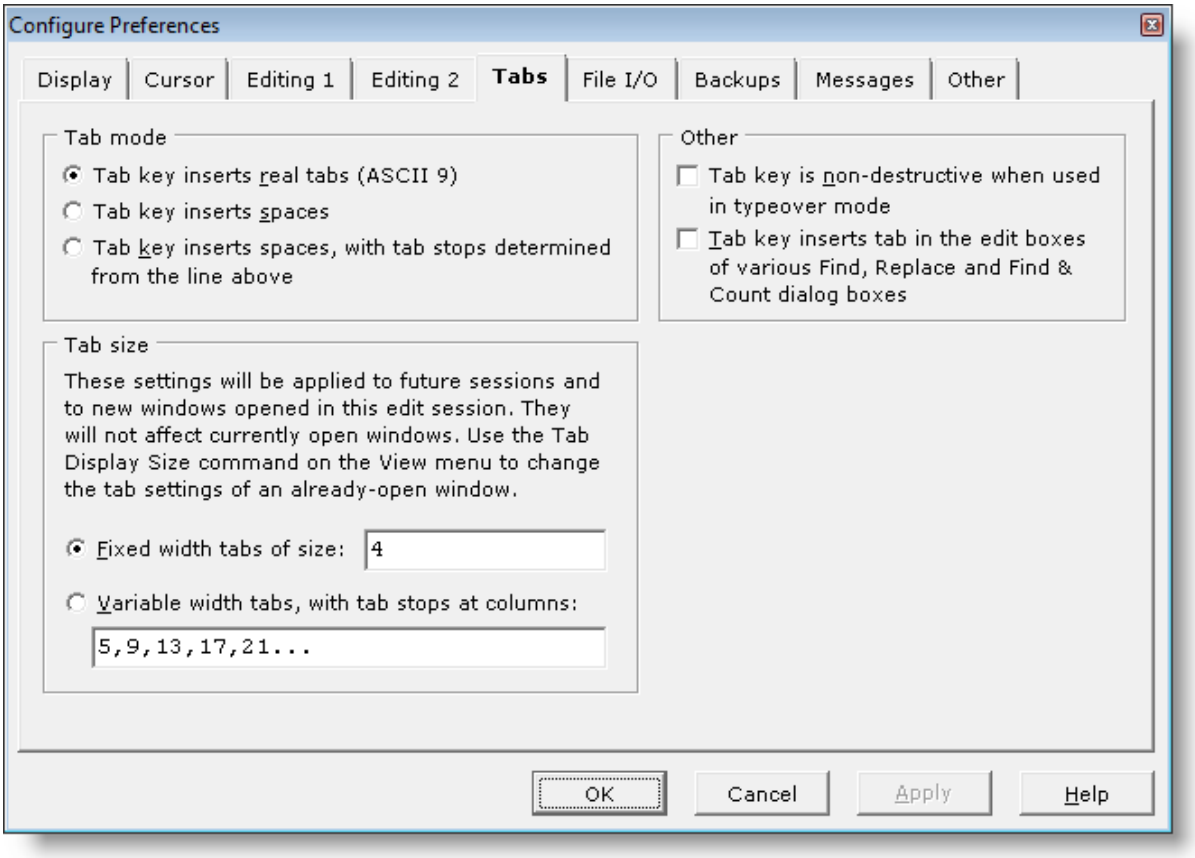
5.196 Preferences - Tabs

Menu: Configure > Preferences

Default Shortcut Key: none

Macro function: ConfigurePreferences()

The Tabs page of the Configure Preferences dialog contains options which relate to function of the *Tab* key, and to the display size of a tab:



Tab mode

Tab key inserts real tabs
When this option is used the *Tab* key will insert tabs (character value 9) into the file.

Tab key inserts spaces
When this option is used the *Tab* key will insert an equivalent number of Spaces, in accordance with the current Tab Display Size.

Tab key inserts spaces, with tabstops determined from line above
When this option is used, the *Tab* key will advance the text cursor to the next field of data as determined from the line above the current line.

Tab Size

Fixed with tabs of size *n*
Use this option to set the default display size for fixed width tabs.

This option sets the display size to be used for tabs in newly created windows, but it does not alter the tab display size of any editor windows that may already be open. The Tab Display Size command on the View menu can be used to set the tab display size for the current file, independent of this default value.

Variable width tabs, with tab stops at columns...

Use this option to designate the columns at which variable width tab stops should occur.

This option sets the tab stops to be used for tabs in newly created windows, but it does not alter the tab stop settings any editor windows that may already be open. The Tab Display Size command on the View menu can be used to set the tab stops for the current file, independent of this default value.

Other

Tab key is non-destructive when used in typeover mode
When this option is checked, the *Tab* key will not overwrite text when used in Typeover mode. For a similar option that affects the function of the *Backspace* key, see the Configure | Preferences | Tabs dialog page.

Tab key inserts tab in the edit boxes of various Find, Replace and Find & Count dialog boxes
When this option is checked, the *Tab* key with insert an actual tab character into the edit boxes of the Find, Replace, Replace Line Enders and Find & Count dialog boxes. Ordinarily, when the *Tab* key is pressed in a dialog box, focus shifts to the next control in the dialog box. This option can be used to override that behavior, making it easier to create search or replace strings that include the tab character.

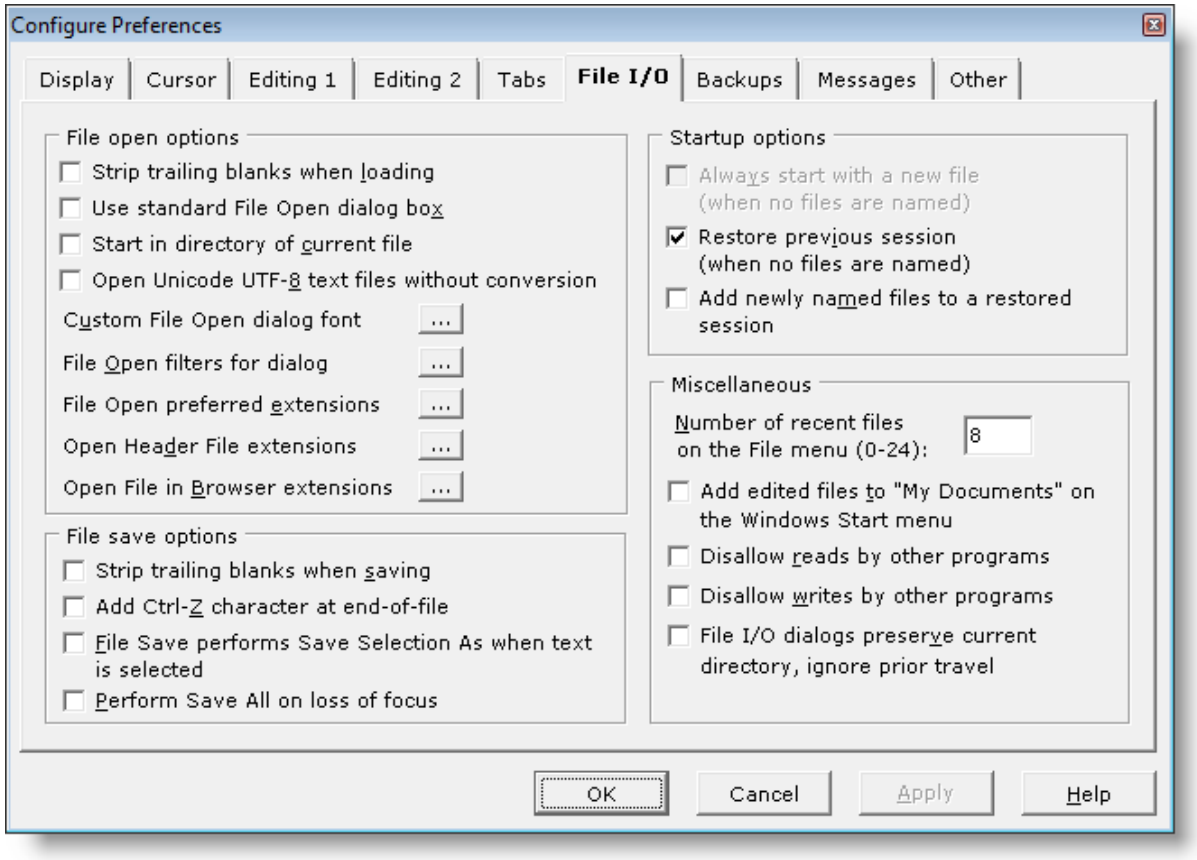
5.197 Preferences - File I/O

Menu: Configure > Preferences

Default Shortcut Key: none

Macro function: ConfigurePreferences()

The File I/O page of the Configure Preferences dialog box contains options which deal with the loading and saving of files:



File Open options

Strip trailing blanks when Loading a file
Use this option to request that trailing Spaces and/or Tabs be removed from the ends of lines as a file is loaded from disk. See also the option titled *Strip trailing blanks when saving a file*.

Use standard File Open dialog box
Use this option to specify that the standard Windows File Open dialog box should be used by the File | Open command. Boxer's custom dialog provides many features that are lacking in the standard dialog. Some users may be more comfortable with the standard dialog, and may wish to select this option.

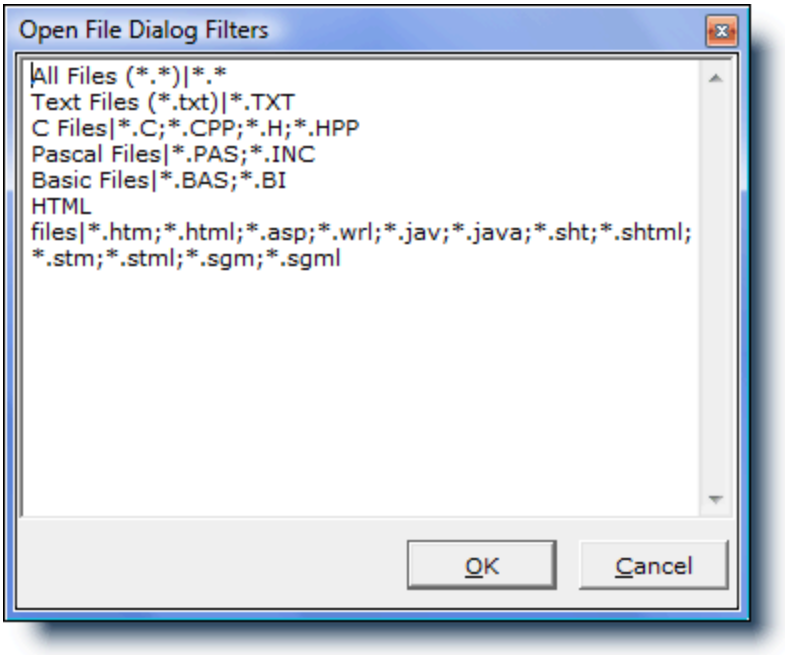
Start in directory of current file
When this option is selected, the File | Open dialog, File | Picker, and Find Text in Disk Files commands will start in the directory of the current file.

Custom File Open dialog font
Use this option to select the font that will be used in Boxer's custom File | Open dialog.

File Open filters for dialog
This option can be used to configure the list of *file filters* which are available from within the File | Open dialog box. File Open filters make it easy to display a group of files: the name of the group can be selected from the drop-down list of file types in the File Open

dialog box.

When the ellipsis (...) button is clicked, a popup editing window will appear which contains the currently defined filters:



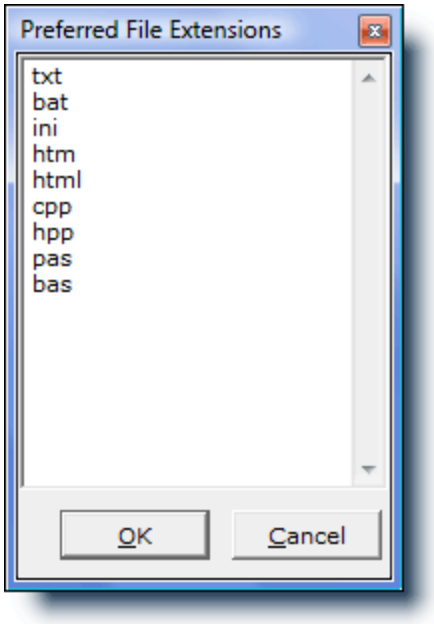
A file filter definition consists of two parts: the *filter name* and the *wildcard file specification*. The filter name contains the text string which will appear in the drop-down list of file types in the File Open dialog box. The wildcard file specification is the expression which is used to match the class of files being defined by the filter.

The filter name is separated from the wildcard file specification with the vertical rule (|) character. When more than one wildcard expression is used, the semi-colon (;) is used to separate the expressions.

File Open preferred extensions
This option can be used to configure Boxer's list of *preferred file extensions*. The list of preferred file extensions is consulted whenever Boxer is asked to open a filename which lacks a file extension.

Before opening a file each extension in the list is added to the supplied filename to see if a file by that name already exists. If so, the file is opened using the extension from the list. If no matching files can be found, the file is opened using the name as originally supplied. In a case where multiple matches might be found, the first matching extension will be used.

When the ellipsis (...) button is clicked, a popup editing window will appear which contains the currently defined extensions:

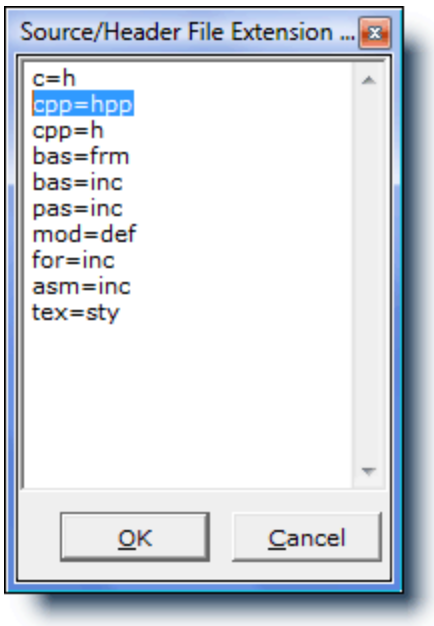


The list can be edited by entering desired extensions into the list, one-per-line. The period (.) should not be included in the extension.

You may need to open a new file named TEST when TXT is a preferred extension, and the file TEST.TXT already exists. In this case, simply specify TEST., with the trailing period, in order to defeat the Preferred File Extension feature.

Open Header File extensions
This option can be used to configure the list of header file extension pairs which are used by the Open Header File command.

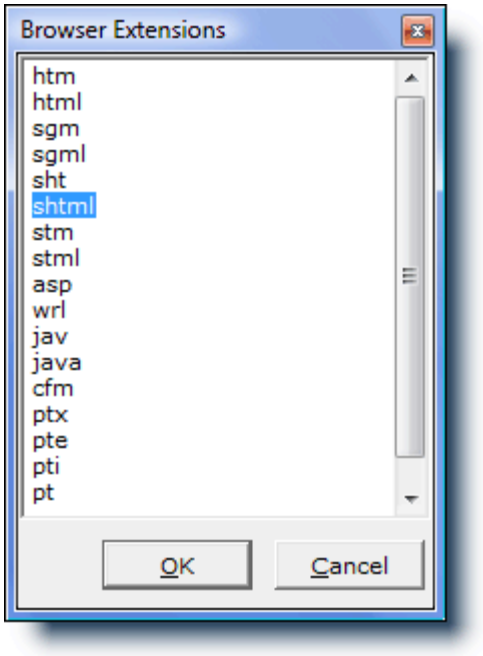
When the ellipsis (...) button is clicked, a popup editing window will appear which contains the currently defined pairs:



Extension pairs are listed one-per-line, with the equal sign (=) being used to separate the extensions. In a case where a file extension has multiple mates, the mate which occurs first in the list will be given priority when attempting to locate the file to be opened.

Open File in Browser extensions
This option can be used to configure the list of file extensions which is used by the Open File in Browser command to determine whether a file is eligible to be opened in an Internet browser.

When the ellipsis (...) button is clicked, a popup editing window will appear which contains the currently defined extensions:



Eligible file extensions are listed one per line. As the HTML standard changes, and as Internet browsers evolve, additional file types will likely become eligible for inclusion in the list.

File Save options

Strip trailing blanks when saving a file
Use this option to request that trailing Spaces and/or Tabs be removed from the ends of lines as a file is saved to disk. See also the option titled *Strip trailing blanks when loading a file*.

Add Ctrl-Z character at end-of-file
This option can be used to request that a Ctrl-Z character--ASCII 26, also known as the end-of-file (EOF) character--be added to a file when saving. Some older programs may require that a file be terminated in this way, but very few modern software packages do.

☞ This option is only applicable when the File Encoding format is set to ASCII; see File Properties for details.

File Save performs Save Selection As, when text is selected
When this option is active and a text selection is present, the Save command will perform the function of the Save Selection As command, saving the *selection* to a specified disk file rather than saving the *file* itself.

Perform Save All upon loss of focus
This option causes Boxer to perform the Save All command whenever focus shifts to another application.

Startup options

Always start with a new file (when no files are named)
Use this option if you prefer that Boxer open a new, untitled file whenever it is launched and another filename is not supplied. This option is not available when the *Restore previous sessions* option is active.

Restore previous sessions (when no files are named)
Use this option if you prefer that Boxer restore the previous edit session whenever it is launched and another filename is not supplied. The restored session will maintain the sizes and positions of all windows, the cursor position in each file, split windows status, and much more. This option is not available when the *Always start with a new file* option is active.

Add newly named files to a restored session
Use this option if you prefer that newly named files be *added* to the edit session which is being restored. This option is not available unless the *Restore previous sessions* option is also active.

Miscellaneous

Number of recent files on the File Menu (0-24)
Use this option to control the number of files which are displayed in the Recent Files list. Up to 24 files can be displayed in this list.


When using Boxer on screens with 800 x 600 resolution it will be necessary to set the number of recent files to four (4) or fewer to prevent the File menu from exceeding the screen height.

Add edited files to 'My Documents' on the Windows Start menu
Use this option to control whether or not files edited by Boxer are added to the *Documents* menu available from the Windows *Start* menu. The Documents menu is preferred by some users as a means to recall previously edited files.

Successful use of this technique requires that the file extension of the file being recalled is 'owned' by the application which last opened the file. This type of 'ownership' is achieved by the use of file associations. By its very nature, a text editor is likely to be called upon to edit many different file types (file extensions). It is probably *not* desirable for a text editor to own the file associations for all the file types it will be asked to edit. Therefore, using the Document menu to launch Boxer will be successful only for file types with which Boxer has been associated.

Disallow reads by other programs
Use this option to request that files which are opened for editing within Boxer be 'locked' so that they cannot be read by other programs. Use of this option will prevent a file from being viewed passively by another program so long as the file is open within Boxer.

Disallow writes by other programs
Use this option to request that files which are opened for editing within Boxer be 'locked' so that they cannot be written to by other programs. Use of this option will

prevent a file from being modified by another program so long as the file is open within Boxer.


A file which is being edited within Boxer could be modified by another program or process. If this condition occurs it will be reported by Boxer as soon as Boxer regains *focus*, and an option will be provided to reload the modified file from disk. An option to disable this option appears on the Configure | Preferences | Messages option page.

File I/O dialogs preserve current directory, ignore prior travel
This option can be used to prevent the various File I/O dialogs (Open, Save, etc.) from changing the record of the current directory due to any directory travel performed from within those dialogs. Ordinarily, the directory last visited within a dialog box is recorded so that it can be used when the dialog next becomes active. This option ensures that directory travel within a dialog box does ***not*** change the record of the current directory.

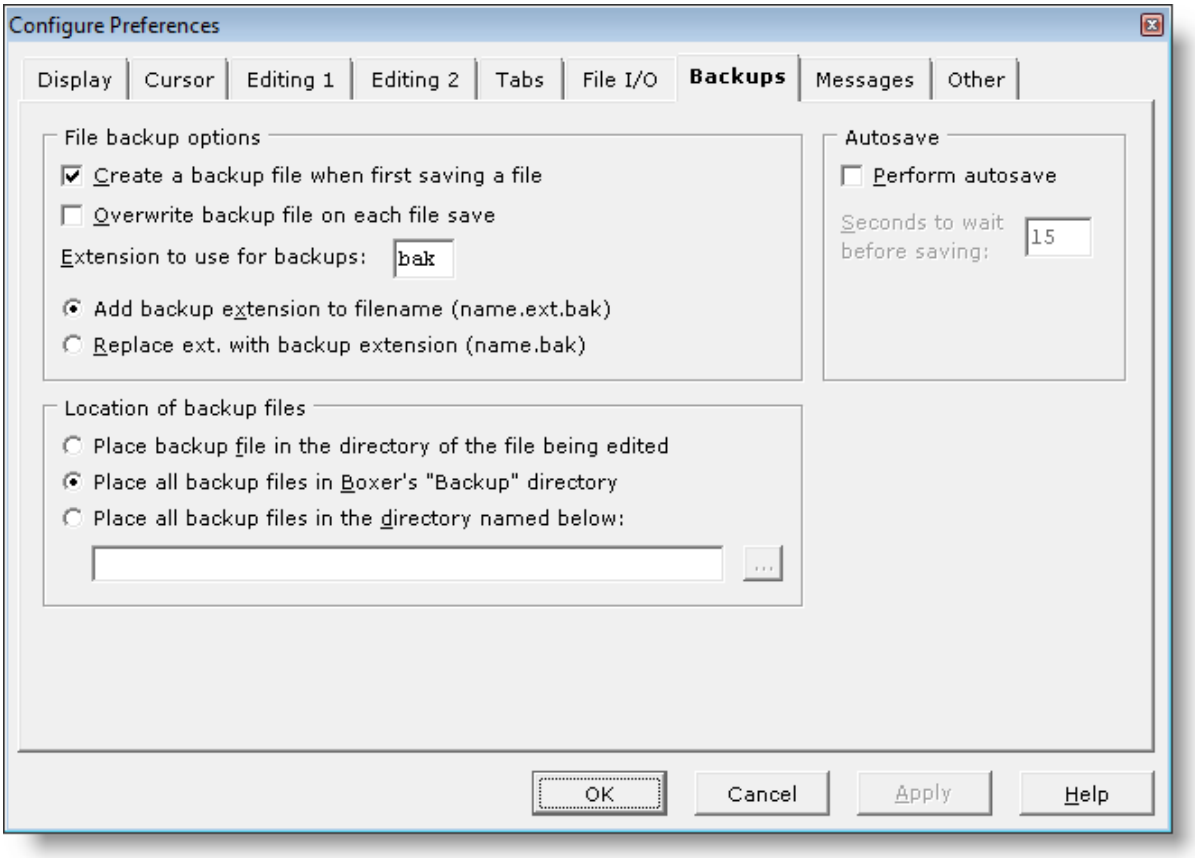
5.198 Preferences - Backups

Menu: Configure > Preferences

Default Shortcut Key: none

Macro function: ConfigurePreferences()

The Backups page of the Configure Preferences dialog box contains options which deal with file backups and the Autosave feature:



File backup Options

Create a backup file when first saving a file
Use this option to request that a backup file be created the first time a file is saved. Subsequent Save operations will not disturb the backup file. If this option is unchecked, all other backup options are disabled.

Overwrite backup file on each file save
Use this option to request that a new backup file be written every time File | Save is performed. When this option is not selected, the backup file will be written the first time a file is saved, but not thereafter.

Extension to use for backups
Use this option to specify the extension which is to be used for file backups. The extension can be 1 to 3 characters long.

Add backup extension to filename (name.ext.bak)
Use this option if you prefer that the backup file extension be *added* to the filename without removing any existing file extension.

Replace extension with backup extension (name.bak)
Use this option if you prefer that the backup file extension *replace* any existing file extension.

Location of backup files

Place backup file in the directory of the file being edited
Use this option to specify that backup files be placed in the same directory as the file being edited. The ellipsis (...) button can be used to browse for a directory using a standard dialog.

Place all backup files in Boxer's "Backup" directory
Use this option to specify that all backup files be placed in the "Backup" directory (folder) which appears in Boxer's home (installation) directory.

Place all backup files in the directory named below
Use this option to specify that all backup files be placed in the directory name which is provided in the associated edit box.


If you frequently edit files of the same name which exist in different directories, you may wish to choose that backup files be kept in the directory of the file being edited. Otherwise, if a common backup directory is used, it's possible that a backup file could be overwritten when later editing a file of the same name from within a different directory. For example, when editing the files `c:\east\sales.txt` and `c:\west\sales.txt`, the file which is saved second would be the one for which the backup file `c:\boxer\Backup\sales.txt.bak` applies.

Autosave

Perform Autosave
This option can be used to indicate that edited files be saved automatically after the supplied time interval has elapsed.


When editing a new, untitled file (see File | New), the Autosave feature will be not be active until the file is saved for the first time, and a filename has been assigned.


When using Autosave on a file that resides on a USB flash drive, it's worth considering the lifetime write limitation of the device. Some USB drives advertise that their lifetime write limitation runs between 10,000 and 1,000,000 uses, so if Autosave were used excessively, it could diminish the life of a USB drive.

Seconds to wait before saving

Use this option to specify the number of seconds after which Autosave should be performed. Caution: using a value which is too small could interfere with data entry when a very large file is being edited, or on very slow PCs.

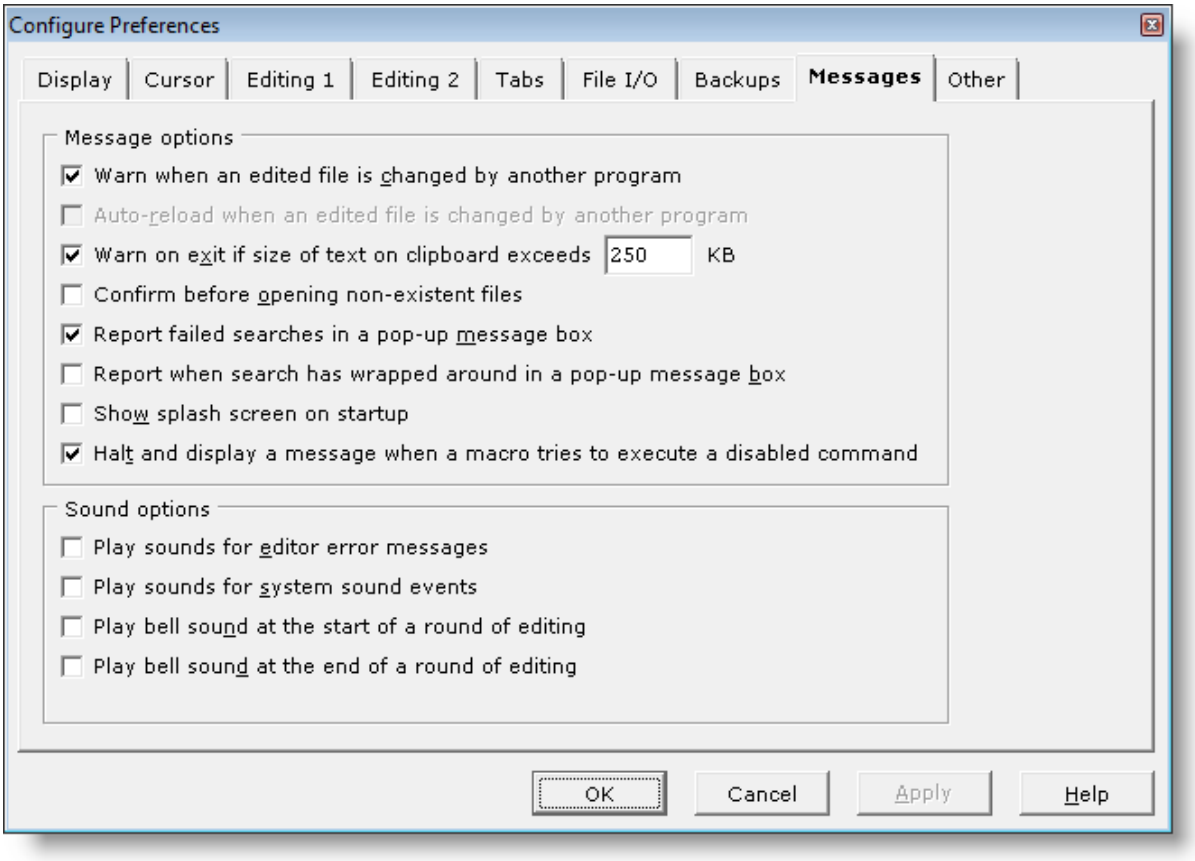
5.199 Preferences - Messages

Menu: Configure > Preferences

Default Shortcut Key: none

Macro function: ConfigurePreferences()

The Messages page of the Configure Preferences dialog box contains options which relate warning messages and sounds.



Message Options

Warn before loading binary files
Use this option if you prefer to be warned before a *binary file* is loaded. Binary files contain the Null character (value 0) and cannot be reliably edited with Boxer. You may wish to use the Open Hex Mode command to view such files in a hex format viewing mode. Boxer will automatically use Open Hex Mode when asked to open files with the extension `.COM`, `.EXE` or `.DLL`.

Warn when an edited file is changed by another program
Use this option if you would like to be notified by Boxer when one of the files you are editing is modified by another program or process. If this option is checked, a dialog box will appear offering a chance to reload the file from disk. If this option is unchecked, notice will NOT be provided when an edited file is modified by another process, and the file will NOT be reloaded..

☞ Proceed carefully in a situation such as this: changes you have made to the file may

be lost when you reload, or saving your file again could overwrite changes which were made by another user.

Auto-reload when an edited file is changed by another program
Use this option if you would like Boxer to automatically reload a file (without issuing a warning) when it senses that it has been changed by another program or process.

☞ Consider carefully the effect of this option: if changes have been made to a file within Boxer and have not been saved, they will be lost if the file is reloaded.

Warn on exit if size of text on clipboard exceeds *n* KB
This option is used to present a warning on exit when the size of the text on the clipboard exceeds the designated threshold. When excessive amounts of text are left on the clipboard, system performance can be impacted.

Confirm before opening non-existent files
This option controls how Boxer reacts when asked to open a file which does not yet exist. If this option is active, a dialog box will be presented to confirm that a new file is to be created. An option is provided to correct a typing error, in case that was the cause for the file not being found. If this option is inactive, Boxer will create a new file using the name provided.

Report failed searches in a pop-up message box
This option controls how Boxer will report a failed search. If this option is active, a popup dialog box will be used to report the failure. Otherwise a message will appear on the Status Bar.

Report when search has wrapped around in a pop-up message box
When Find Next or Find Previous are used in wrap around mode, a message appears on the status bar when the search has wrapped back to the location of the first match. Use this option if you prefer that this event be reported in a message box instead.

Show splash screen on startup
This option controls whether or not Boxer's splash screen graphic will be displayed on-screen while the program initializes. Display of the splash screen graphic can also be controlled using the -G command line option flag.

Halt and display a message when a macro tries to execute a disabled command
This option is intended for advanced users. By default, Boxer will prevent macro functions from being executed when the underlying command being run is disabled within the menus. Commands are disabled when the environment is unsuitable for them to be run. Advanced users may wish to override this behavior, if they think they have reason to disregard Boxer's disabling of a given command.

Sound Options

Play sounds for editor error messages
Use of this option causes the system sound for *Default Beep* to be played for editor errors.

Play sounds for system sound events

Use of this option causes the system sounds for *Minimize*, *Maximize*, *Restore Up* and *Restore Down* to be played when these events occur.

☞ The sounds which are used for various system sounds can be defined from Start Menu | Settings | Control Panel | Sounds.

Play bell sound at the start of a round of editing
Use of this option causes Boxer to play its two-bell sound to signal the beginning of a new round of editing.

Play bell sound at the end of a round of editing
Use of this option causes Boxer to play its one-bell sound to signal the end of a round of editing.

☞ If your computer is not equipped with a sound card, Boxer will not be able to play the sounds described above.

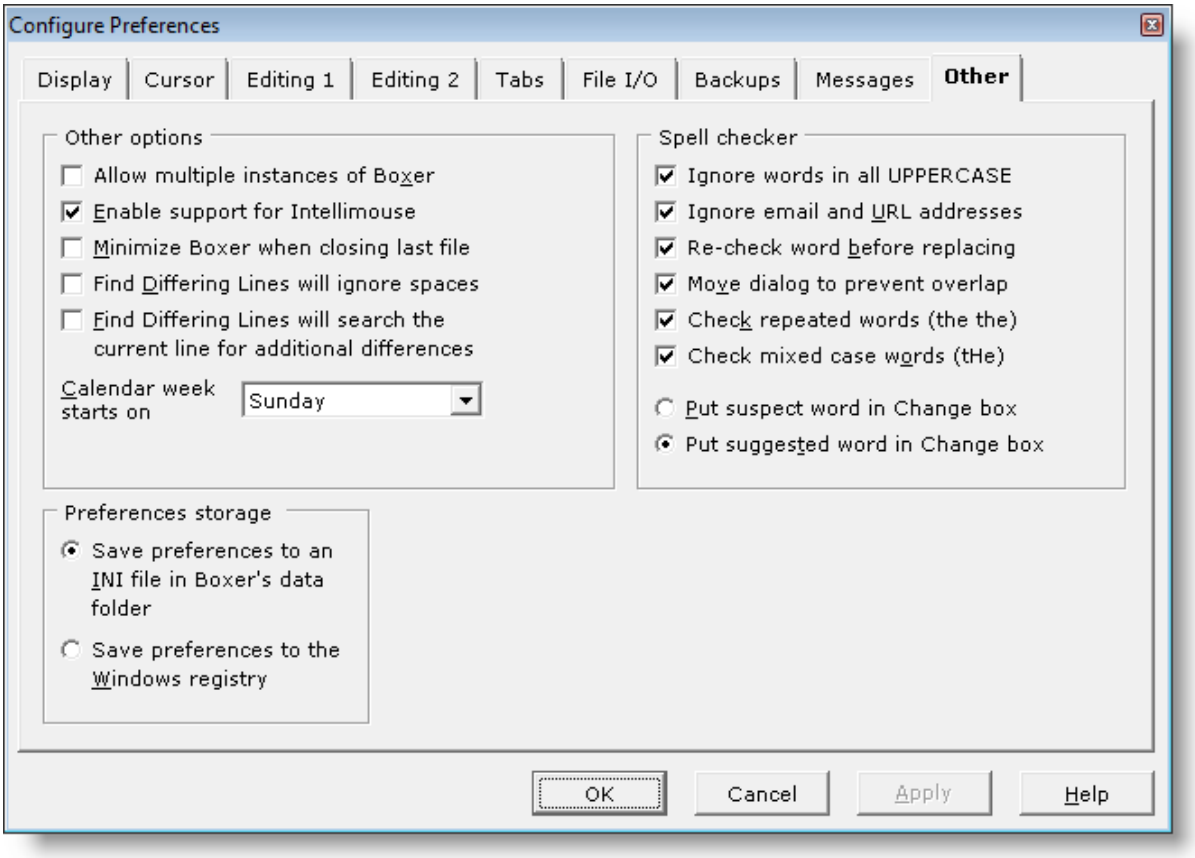
5.200 Preferences - Other

Menu: Configure > Preferences

Default Shortcut Key: none

Macro function: ConfigurePreferences()

The Other page of the Configure Preferences dialog box contains options which do not logically group with its other pages. Among these are startup options, Spell Checker options and miscellaneous options:



Other options

Allow multiple instances of Boxer
Use this option to control whether or not multiple instances of Boxer are allowed to run concurrently. When multiple instances are allowed, a new Boxer session will be launched each time the program is run. When multiple instances are disallowed, each would-be instance of Boxer will pass the files named for editing to the session which is already running. If files are not named for editing the existing session will simply become current.

☞ **Caution**: when multiple instances of Boxer are run concurrently confusion can arise regarding the editor's configuration settings. Boxer writes its configuration information to the *Windows registry* each time it exits. When multiple instances of Boxer are run the configuration settings from the first sessions that are exited will be overwritten by any sessions which exit later on. In essence, the configuration settings from the last exited Boxer session will be the settings which are ultimately recorded in the registry.

Enable support for Intellimouse
Use this option to enable support for the Microsoft Intellimouse device. The Intellimouse has a *mousewheel* which permits a document to be scrolled without the use of the Vertical Scroll Bar. The mousewheel also doubles as a center mouse button, and can therefore be used to perform columnar text selections.

Minimize Boxer when closing last file
When this option is active, Boxer will minimize itself to the *task bar* when its last file is closed. Click on the Boxer button in the task bar to restore the application.

Find Differing Lines will ignore spaces
Use this option to force the Find Differing Lines command to ignore leading and trailing Spaces and Tabs when comparing lines. Lines which differ only in their indent, or due to trailing *whitespace*, will be considered equal.

Find Differing Lines will search the current line for additional differences
Use this option to instruct the Find Differing Lines command to continue search along the current line for differences after the first difference has been reported. A manual re-syncing of the text cursor position may be required.

Word/Sentence/Title case commands will convert text to lowercase before operation
When applying Word, Sentence or Title case to a text selection, the question arises whether or not the text should first be forced to lowercase before the operation is performed. When this checkbox is checked, the case of the selected text *will* be adjusted before applying the requested conversion. You may wish to review the text after conversion to ensure that proper nouns, acronyms, and other capitalized words have been properly converted.

Calendar week starts on
Use this option to designate the day of week which should be used to start a week in the pop-up Calendar. The default setting is Sunday, but users in some countries will prefer a different setting.

Preferences Storage

Save preferences to an INI file in Boxer's data folder
When this option is selected, Boxer will save its preferences to a disk-based file named `BOXER.INI` which is located in its *data folder*. For more information, see Portable Editing.

Save preferences to the Windows registry
When this option is selected, Boxer will save its preferences to the Windows registry. The location used is:

```
HKEY_CURRENT_USER/Software/Boxer Software/Boxer Text Editor NN
```

where 'NN' represents the current major version number.

Spell Checker

Ignore words in all UPPERCASE
Use this option to indicate that the Spell Checker should ignore words which appear in uppercase during its operation. This helps prevent false reports when spell checking files which contain *acronyms* or filenames.

Ignore Email and URL addresses
Use this option to indicate that the Spell Checker should ignore email and *URL* addresses during its operation.

Recheck word before replacing
Use of this option causes the Spell Checker to recheck a replacement word before making the change. This provides a double check when you elect to type a replacement word rather than using one from the supplied list.

Move dialog to prevent overlap
Use this option to request that the Spell Checker position its dialog box so as not to obscure the context of the suspect word which is being reported. The position of the mouse cursor is moved along with the dialog, so you won't need to 'chase' the dialog around the screen throughout a spell checking session.

Check Repeated Words
Use this option to request that the Spell Checker watch for repeated words, such as 'the 'the'. When an offending sequence is found, an option will be provided to delete the duplicated word.

Check Mixed Case Words
Use this option to request that the Spell Checker check the spelling of mixed case words. Mixed case words can occur due to a typographical error, or when an acronym (GmbH) or company name (SoftSeek) is being used.


Due to a limitation in the way the dictionary vendor stores entries in its user dictionary, it is not possible to add a legitimate mixed case word (such as eBay) so that future alerts for that word will not occur.

Put suspect word in Change box
Use this option if you prefer that the suspect word be placed in the Change edit box.

Put suggested word in Change box
Use this option if you prefer that the suggested correction be placed in the Change edit box.

5.201 Previous Bookmark

Menu: Jump > Previous Bookmark

Default Shortcut Key: Shift+Ctrl+Up

Macro function: PreviousBookmark()

The Previous Bookmark command moves the text cursor to the nearest bookmark which appears above the text cursor's current location. If the Previous Bookmark command finds no bookmarks above the current line, the text cursor will wrap around and be

placed on the last bookmark in the file.

Travel among bookmarks is based upon location, not bookmark number.

The Bookmark Manager can be used to view all bookmarked lines in a single view, and navigate to, or delete, selected bookmarks.

Bookmarks will persist for the current editing session, and will be restored when restoring an edit session.

If a selection exists when this command is issued, the selection will be extended to the bookmarked location.

5.202 Previous Function

Menu: Jump > Previous Function

Default Shortcut Key: Ctrl+Alt+Up

Macro function: PreviousFunction()

The Previous Function command moves the cursor to the previous function (or procedure) declaration within the current file. This command makes it possible to move backward through a source code file on a function-by-function basis.

If a text selection is present when this command is issued, the selection will be extended to the new cursor location.

The Previous Function command relies upon the Ctags Function Index feature to perform its service. If the Ctags feature has been disabled, or if the file being edited is not supported by Ctags, the Previous Function command will be unavailable.

The types of Ctags identifiers for which this command applies is user-configurable. The default setting includes entries for "function", "procedure", "subroutine", "method", etc. The full list can be viewed or changed on the *Advanced* tab of the Configure | Ctags Function Indexing dialog.

5.203 Previous Paragraph

Menu: Jump > Previous Paragraph

Default Shortcut Key: none

Macro function: PreviousParagraph()

The Previous Paragraph command moves the text cursor to the start of the previous paragraph.

For purposes of this command, a paragraph is considered to be a block of lines with one or more empty lines between them. Contiguous paragraphs which are denoted by a change of indent on the first line, and not by an intervening blank line, will not be recognized to be distinct paragraphs.

If a text selection is present when this command is issued, the selection will be extended to the new cursor location.

5.204 Print

Menu: File > Print > Print Normal / Print Color Syntax / Print Mono Syntax

Default Shortcut Key: Ctrl+P (Print Normal)

Macro functions: Print / PrintColor / PrintMonochrome

The Print command is used to send the current file to the printer. The layout of the printed page will be determined by the current settings within the Page Setup dialog. You may wish to use Print Preview to display the print job on-screen before sending it to the printer.

Printing can be performed in any of three modes:

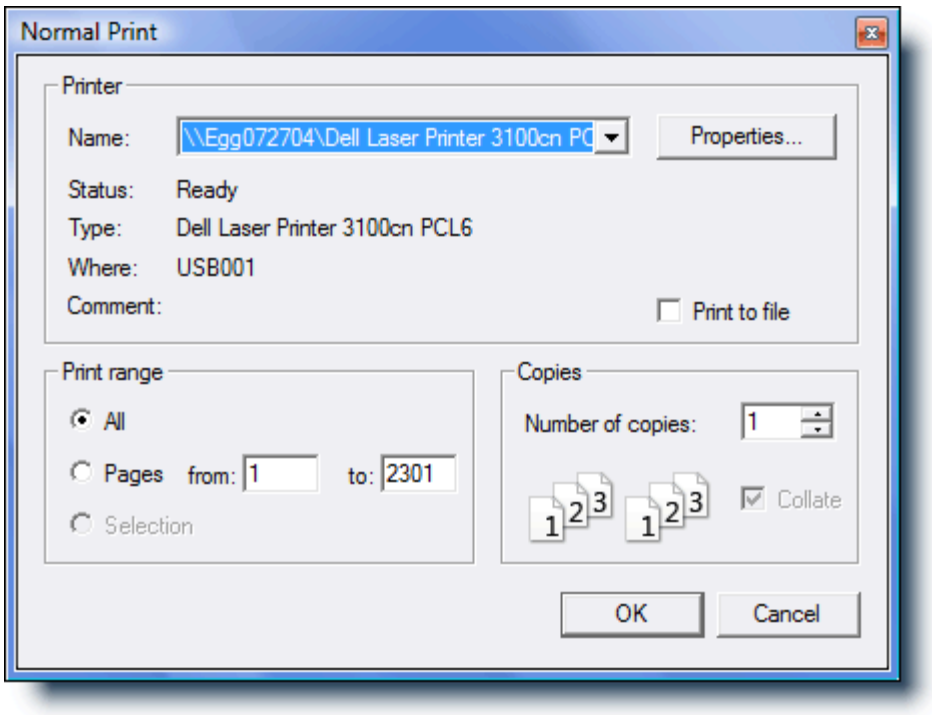
Normal
The file is printed without applying syntax highlighting coloration.

Color Syntax
The file is printed with color syntax highlighting applied. Syntax elements will be displayed in color, and in bold, italic and/or underlined font styles in accordance with the current settings.

Monochrome Syntax
The file is printed with monochrome syntax highlighting applied. Syntax elements will be displayed in bold, italic and/or underlined font styles in accordance with the current settings.

The Monochrome Syntax and Color Syntax printing modes will be disabled when the file being edited is a file type for which Syntax Highlighting information is not available.

The standard Windows print dialog is presented before printing begins:



From here you can select the range of pages to be printed, and the number of copies to be printed. If text has been selected, you will be able to choose a radio button which dictates that only the selected text is to be printed. See the note below regarding printing selections of program source code.

Boxer will automatically convert any Tabs within the text being printed to Spaces. This ensures that the Tab Display Size used by Boxer does not conflict with that of the printer.

The color/font settings for Monochrome Syntax Print and Color Syntax Print are accessed from the Configure | Colors dialog by selecting the appropriate mode from the drop-down list at the upper left.

☞ When printing selected text from program source code using either Monochrome or Color Syntax Print mode, improper highlighting can occur if the starting or ending points of the selection fall in the middle of a syntax element, or if the starting line of the selection falls within a multi-line comment block. Likewise, printing a columnar selection is likely to result in improper syntax highlighting. For best results, choose the selected area logically so that its start and end points occur at natural break points in the code.

☞ Strictly speaking, the use of Color Syntax Printing requires that a color printer be installed and attached. Some users may wish to use the *Print to File* option available from the Print dialog to save a print image on disk at one PC for later printing on another PC with a color printer. For this reason the Print Color Syntax command is not disabled when the PC is found to lack a color printer. Users will need to be careful when transporting a printer image file in this way, because it may not be compatible with the destination printer.

A formfeed character can be placed in column one--or in the last position on a line--to indicate that a new page should begin at that point. The footer of the page--if one has been defined--will be printed and the page will eject. A formfeed in any other column will be ignored by Boxer's print routine.

5.205 Print All

Menu: File > Print > Print All Normal / Print All Color Syntax / Print All Mono Syntax

Default Shortcut Key: none

Macro functions: PrintAll / PrintAllColor / PrintAllMonochrome

The Print All command is used to send all files currently being edited to the printer. The layout of the printed page will be determined by the current settings within the Page Setup dialog. You may wish to use Print Preview to display the print jobs on-screen before sending it to the printer.

Printing can be performed in any of three modes:

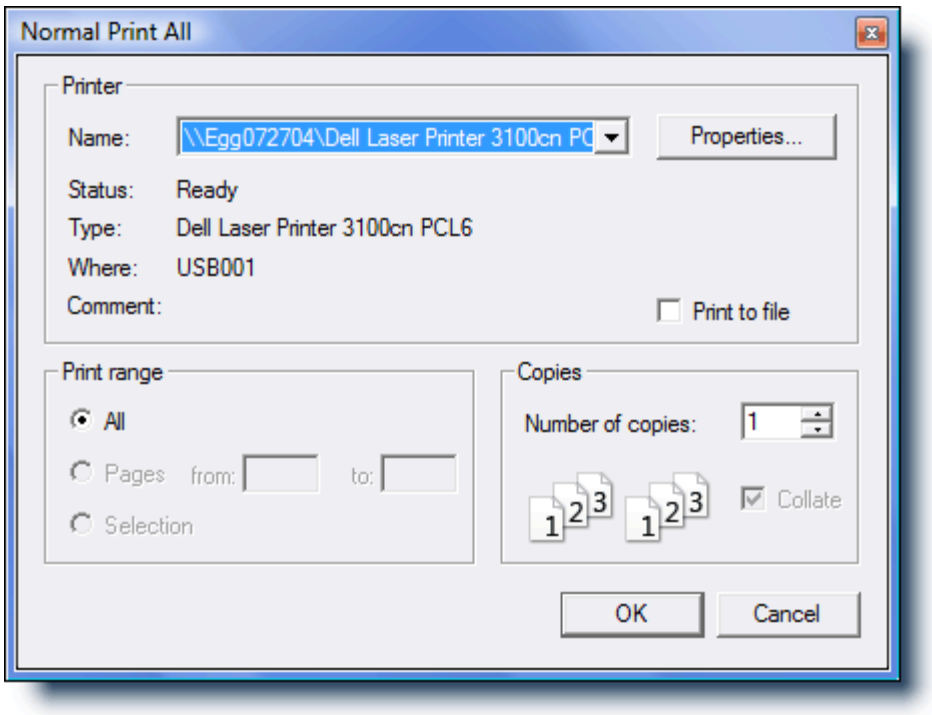
Normal
The file is printed without applying syntax highlighting coloration.

Color Syntax
The file is printed with color syntax highlighting applied. Syntax elements will be displayed in color, and in bold, italic and/or underlined font styles in accordance with the current settings.

Monochrome Syntax
The file is printed with monochrome syntax highlighting applied. Syntax elements will be displayed in bold, italic and/or underlined font styles in accordance with the current settings.

The Monochrome Syntax and Color Syntax printing modes will be disabled when the file being edited is a file type for which Syntax Highlighting information is not available.

The standard Windows print dialog is presented before printing begins:



From here you can select the range of pages to be printed, and the number of copies to be printed. If text has been selected, you will be able to choose a radio button which dictates that only the selected text is to be printed. See the note below regarding printing selections of program source code.

Boxer will automatically convert any Tabs within the text being printed to Spaces. This ensures that the Tab Display Size used by Boxer does not conflict with that of the printer.

The color/font settings for Monochrome Syntax Print and Color Syntax Print are accessed from the Configure | Colors dialog by selecting the appropriate mode from the drop-down list at the upper left.

☞ When printing selected text from program source code using either Monochrome or Color Syntax Print mode, improper highlighting can occur if the starting or ending points of the selection fall in the middle of a syntax element, or if the starting line of the selection falls within a multi-line comment block. Likewise, printing a columnar selection is likely to result in improper syntax highlighting. For best results, choose the selected area logically so that its start and end points occur at natural break points in the code.

☞ Strictly speaking, the use of Color Syntax Printing requires that a color printer be installed and attached. Some users may wish to use the *Print to File* option available from the Print dialog to save a print image on disk at one PC for later printing on another PC with a color printer. For this reason the Print Color Syntax command is not disabled when the PC is found to lack a color printer. Users will need to be careful when transporting a printer image file in this way, because it may not be compatible with the destination printer.

A formfeed character can be placed in column one--or in the last position on a line--to indicate that a new page should begin at that point. The footer of the page--if one has been defined--will be printed and the page will eject. A formfeed in any other column will be ignored by Boxer's print routine.

5.206 Print Preview

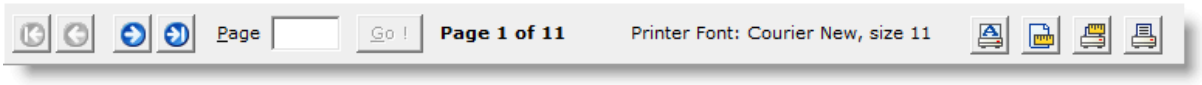
Menu: File > Print Preview > Normal / Color Syntax / Mono Syntax

Default Shortcut Key: none

Macro functions: PrintPreview, PrintPreviewMonochrome, PrintPreviewColor

The Print Preview command provides a means of viewing a print job on-screen before it is sent to the printer. The print document will be shown in a window that has controls for moving quickly to any page within the document. The *PgUp* and *PgDn* keys can also be used to page through the document. You can click with the left mouse in the turned page corner to advance to the next page; clicking with the right mouse button will move backward to the previous page.

The Print Preview form has buttons to access the Printer Font, Page Setup, Print Setup and Print commands directly from Print Preview mode. This makes it easy to see the effect of a font face or size change, or to view the result of changing margins, paper orientation, header and footer text, line spacing, line numbering or line wrapping.



Print Preview can be performed in any of three modes:

Normal
The preview is shown without the application of syntax highlighting coloration.

Monochrome Syntax
The preview is shown with monochrome syntax highlighting applied. Syntax elements will be displayed in bold, italic and/or underlined font styles in accordance with the current settings.

Color Syntax
The preview is shown with color syntax highlighting applied. Syntax elements will be displayed in color, and in bold, italic and/or underlined font styles in accordance with the current settings.

The Monochrome Syntax and Color Syntax preview modes are disabled unless the file being edited is a file type for which Syntax Highlighting information is defined.

The color and font style settings for Monochrome Syntax and Color Syntax are accessed from the Configure | Colors dialog by selecting the appropriate mode from

the drop-down list at the upper left.

Because of the difference in resolution between the printer and the screen, the screen can never present a perfect image of the printed page. Under some circumstances, with certain font sizes, you may see imperfect line or character spacing or other small inaccuracies. It should not be assumed that the printed page will have the same inaccuracies. Print Preview can be expected to accurately show the layout of the page, page breaks, positioning, etc.

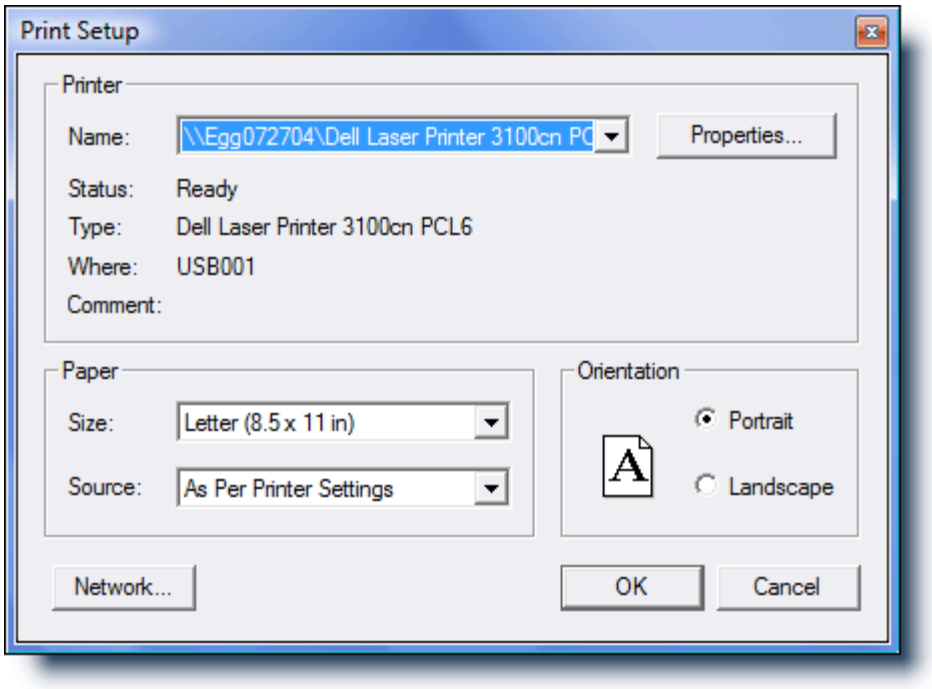
5.207 Print Setup

Menu: File > Print Setup

Default Shortcut Key: none

Macro function: PrintSetup()

The Print Setup command provides access to the standard Windows print setup dialog which allows the current printer to be selected. You can also select paper size, paper orientation (*portrait* or *landscape*), and other options which may vary from printer to printer.



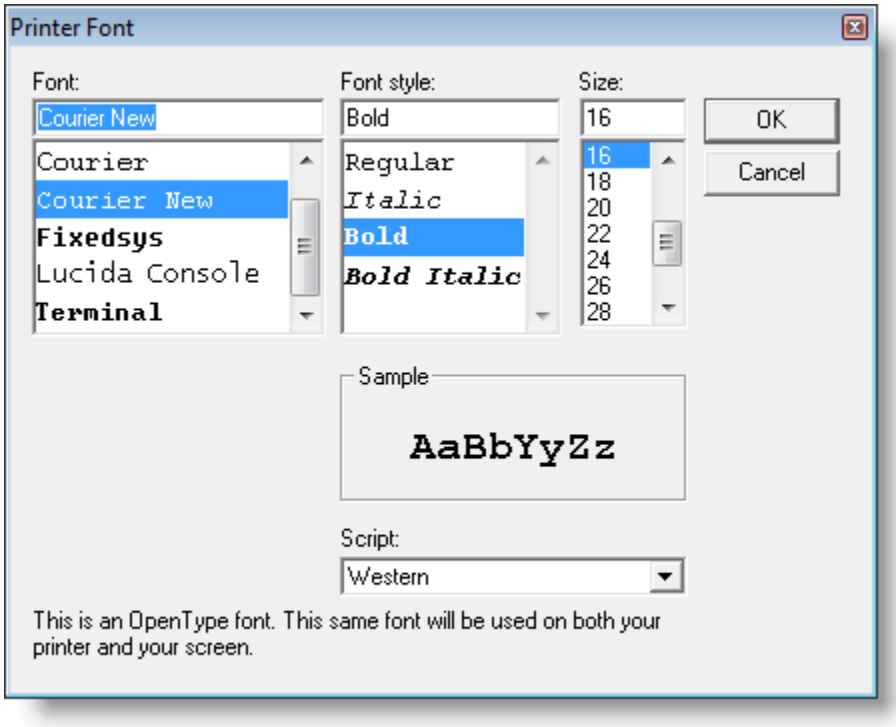
5.208 Printer Font

Menu: Configure > Printer Font

Default Shortcut Key: none

Macro function: ConfigurePrinterFont()

The Printer Font command is used to select the font that is used to print files from within Boxer. The standard Windows font dialog is presented for selecting the screen font:



Boxer requires that *fixed width* fonts be used, so the Printer Font dialog box does not display *proportionally spaced* fonts. This is required, in part, to ensure that columnar selections can be highlighted neatly in rectangular blocks, and so that the Column Ruler can be used. These features would not be possible if the use of proportional fonts were permitted.

The *Font* listbox at the left of the dialog displays the *fixed width* fonts which are available for selection.

The *Font Style* listbox display the styles which are available for the selected font. Typically these are *Regular*, *Italic*, *Bold* and *Bold Italic*, although some fonts may not offer all styles.

The *Size* listbox displays the sizes which are available for the selected font.

The *Script* drop-down list displays the various character mappings which are available for the selected font.

The colors which are used when using Color Syntax Printing are controlled via the Configure Colors command.

When selecting a True Type font, the standard Windows font dialog box may display a message at the bottom indicating that the selected font will be used for both screen display and printing. This message does *not* apply to the use of fonts within Boxer, since Boxer permits the Screen Font and Printer Font to be selected separately. The message intends to convey the idea that the font is *capable* of being used for both the screen and the printer.

5.209 Quote and Reformat

Menu: Paragraph > Quote and Reformat

Default Shortcut Key: Ctrl+Q

Macro function: QuoteAndReformat()

The Quote and Reformat command can be used to reformat a paragraph within the defined Text Width and according to the current Justification Style, while adding a quoting symbol to the left edge of the paragraph. This formatting style is used within email replies and in other communications to visually identify the text which is being replied *to* from the text of the reply itself.

Two quoting styles are available: one in which the first line is quoted and additional lines are indented to match the first line:

```
>> A  Multi-User  License  provides an inexpensive
   way for businesses,  schools,  universities  or
   other  work  groups  to  supply their personnel
   with computer software in both a legal and cost
   efficient manner.  By  licensing  Boxer for use
   on multiple computers you can standardize on  a
   single  editing  tool that will serve the needs
   of all people within the group.
```

and one in which all lines within the paragraph are quoted:

```
>> In so doing,  support and maintenance costs can
>> be reduced, and users can benefit  from  having
>> ready  access  to others who are using the same
>> software.  Multi-user  licensing  is  also more
>> economical than  making  individual  purchases,
>> because there is no need for us to supply extra
>> disks, reference literature, etc. for all users
>> within the group.
```

Both the quoting style, and the quoting symbol used, can be configured on the Configure | Preferences | Editing 1 options page.

The Quote and Reformat command makes use of the Reformat command internally during its operation. As noted in the Reformat command, lines beginning with a period (.) are treated as blank lines in order to recognize text markup tags. As a result, the use of a quoting string that begins with a period will not produce the

desired results, and should be avoided. All other symbols and characters are permissible.

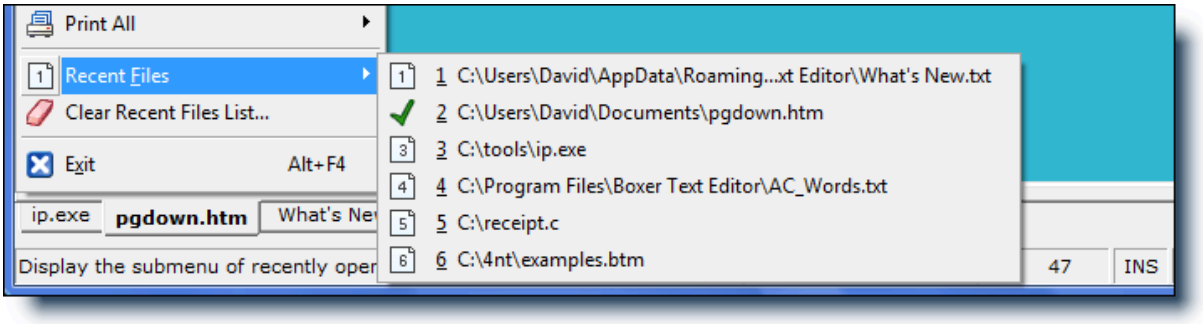
5.210 Recent Files

Menu: File

Default Shortcut Key: not applicable

Macro function: OpenRecentFile()

The Recent Files list appears near the bottom of the File menu, above the Exit command. Each time a file is opened for editing, its name is added to the list. If necessary, the eldest entry is bumped from the list. This list makes it easy to recall files which were recently viewed or edited without the need to use the Open command, as is typically done. The filenames are displayed with a 'hot' number to their left, so that *Alt+F*, *F*, followed by the number, will load the named file.



The number of files displayed in the list can be controlled on the Configure | Preferences | File I/O options page. The option is named *Number of recent files on the File menu*.

Long filenames will be shortened for display if the relevant option on the Configure | Preferences | Display options page is checked.

5.211 Recent Projects

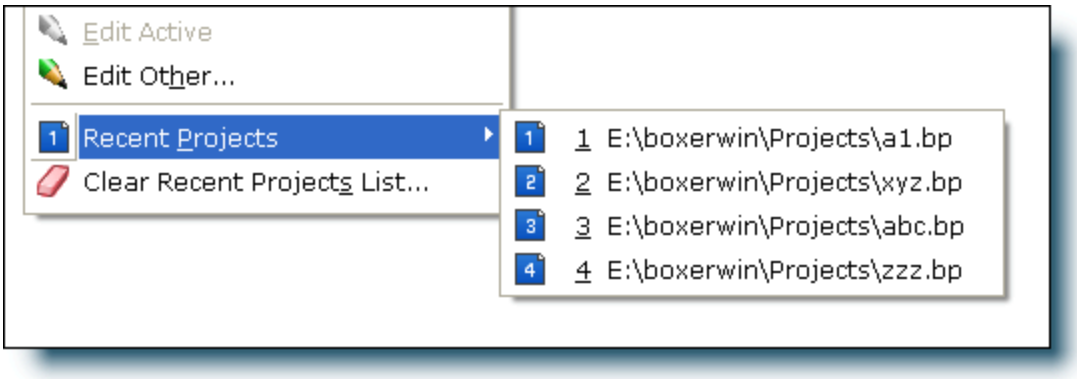
Menu: Project

Default Shortcut Key: not applicable

Macro function: OpenRecentProject()

The Recent Projects list appears near the bottom of the Projects menu. Each time a projects is opened, its name is added to the list. If necessary, the eldest entry is bumped from the list. This list makes it easy to recall projects which were recently opened without the need to use the Project | Open command. The projects are

displayed with a 'hot' number to their left, so that *Alt+P*, *P*, followed by the number, will load the named project.



Up to 16 recent projects can appear on the Recent Projects list.

Long project names will be shortened for display if the relevant option on the Configure | Preferences | Display options page is checked.

5.212 Record Keys

Menu: Tools > Record Keys

Default Shortcut Key: Shift+F5

Macro function: none

Use the Record Keys command to begin recording a series of keystrokes for later playback with the Playback Keys command. This command provides a *'macro-by-example'* or *'on-the-fly'* keystroke recording facility. While recording is in process, all keystrokes entered from within the editor window will be stored. Mouse motion is not recorded *per se*, but if the mouse is used to select a command from the main menu, that command will be recorded just as though its shortcut key had been used instead. To stop recording, simply issue this command again: its name in the main menu changes to *Stop Recording* while recording is in process.

Mouse and/or keystroke interaction with popup dialogs is not stored in a key recording. If a confirmation dialog appears during keystroke recording, the response to that dialog is *not* recorded. The dialog will appear during playback, and will need to be dismissed manually at run-time.

If you need to pause in the middle of a keystroke recording, use the Pause Recording command. This will allow you to perform operations in the editor that will not appear in the key recording. Issue the command again to resume recording.

Command key assignments are not used in the files created by Save Key Recording,

so there's no chance that changing key assignments might disturb an existing key recording. Likewise, key recording files can be freely shared with other Boxer users without concern that playback might be influenced by differing key assignments on the target machine. However, depending on the nature of the key recording, default settings on the target machine (autoindent, typing wrap, etc) could influence how a key recording performs upon playback.

☞ There are several system-reserved key sequences that cannot be used within a keystroke recording. These key sequences all relate to moving among or closing document windows. The operating system traps these key sequences before they're ever seen by the application. Boxer provides its own key assignments for these commands, and they're available from the main menu as well, so it's not the *functionality* that need not be avoided. Rather, the *method of activating* these functions might need to be altered when making a recording. The key sequences to be avoided are the following: Ctrl+Tab, Shift+Ctrl+Tab, Ctrl+F4, Shift+Ctrl+F4, Ctrl+F6, Shift+Ctrl+F6.

☞ The Auto-Complete feature is disabled during keystroke recording. Since the entries that are presented in the pop-up Auto-Complete suggestion list are sensitive to the file being edited, and to text that has been previously typed, it's not safe to assume that the order of the list will be the same at playback time.

☞ If you need to automate a repetitive task in which logic must be applied to the recording, use Boxer's full-powered Macro language.

5.213 Redo

Menu: Edit > Redo

Default Shortcut Key: Ctrl+Y

Macro function: Redo()

The Redo command can be used to reverse the effect of the most recent Undo command. For example, while issuing a series of Undo commands to reverse unwanted changes, you suddenly see that the last Undo went too far. Issuing Redo will undo the last Undo. Undo and Redo are opposites.

Undo cannot be undone after additional changes are made; Redo must be issued before other changes are made.

The Redo command is disabled until at least one Undo command has been performed.

5.214 Redo All

Menu: Edit > Redo All

Default Shortcut Key: none

Macro function: RedoAll()

The Redo All command can be used to undo the effect of all Undo commands which have been issued since the last change was made. Unless you feel sure about the number of undos which have been recorded by Redo, it is often safer to use the Redo command to step singly through the changes so that their effect can be seen on-screen before proceeding.

5.215 Reference

Menu: Jump > Reference

Default Shortcut Key: none

Macro function: Reference()

The Reference command is used in conjunction with the Declaration command. After issuing the Declaration command to jump from an identifier reference to its declaration, use the Reference command to return to the point of reference.

5.216 Reformat

Menu: Paragraph > Reformat

Default Shortcut Key: Ctrl+F10

Macro function: Reformat()

The Reformat command can be used to reformat the paragraph at the text cursor within the defined Text Width and according to the current Justification Style. The Reformat operation begins on the current line and includes all lines to the end of the current paragraph (see note below). The text cursor is advanced to the first line of the next paragraph following Reformat, so that successive Reformat commands will move smoothly through the document.

If a range of lines is selected, all paragraphs within the selected range will be reformatted. Use the Select All Text command before Reformat to reformat an entire file, but first check to be sure that the file doesn't contain tables or lists which might be adversely affected by reformatting.


When the Reformat command reformats text, it does so by adding a newline (hard line ender) at the end of the line. To wrap text visually, without introducing a hard line ender into the file, see the Visual Wrap command.

Fully Indented Paragraphs
Boxer uses the amount of indent on the second line of the paragraph to determine the indent level for the entire paragraph. A paragraph can be made *fully indented* by manually indenting the first and second lines of the paragraph and then reformatting.

```
    This paragraph is fully indented. This
    paragraph is fully indented. This paragraph
    is fully indented. This paragraph is fully
    indented. This paragraph is fully indented.
    This paragraph is fully indented.
```

Hanging Indents
The indent on the first line of the paragraph is not applied to other lines in the paragraph. A *hanging indent* can be achieved by placing less indent on the first line of the paragraph than on the second line. Likewise, if the first line of a paragraph has extra indent, it too will be preserved.

```
This paragraph has a hanging indent. This paragraph
    has a hanging indent. This paragraph has a
    hanging indent. This paragraph has a hanging
    indent. This paragraph has a hanging indent. This
    paragraph has a hanging indent.
```

Bulleted Paragraphs
If you wish to create bulleted paragraphs that will retain their layout after being reformatted, use a Tab character to separate the bullet from the body of the paragraph. If a series of Spaces were to be used between the bullet and the body text they would be adjusted during Reformat. Tabs are maintained in this situation.

```
*   This paragraph uses a bullet separated from the
    body text with a Tab character. This paragraph
    uses a bullet separate from the body text with a
    Tab character.
```

 To have text wrap to the next line automatically as you type, use the Typing Wrap feature. To wrap text visually, use the Visual Wrap command.

 The Unformat command is essentially the opposite of Reformat: it removes line enders from a paragraph to create a long, flowing line of text.

 The end of a paragraph is signaled by the presence of one or more blank lines between paragraphs, but *not* simply by the presence of additional indent on a line which immediately follows the current paragraph. Lines which begin with a period (.) will also be recognized as blank lines for purposes of Reformat. This is to permit the use of text markup languages such as Flexicon for adding formatting commands to text.

 Reformat will break lines between HTML or XML tags, when appropriate, even if an intervening space is not present.

5.217 Regular Expressions

When searching for a text string using the Find, Replace, Replace Line Enders or Find

Text in Disk Files commands, Boxer supports the use of Regular Expressions, a pattern matching grammar first popularized on the Unix operating system. Regular Expressions make it possible to specify a search string which can match many different target strings, or to restrict the ways in which a search string can be matched.

Boxer uses Perl-Compatible Regular Expressions as implemented by the increasingly popular PCRE 5.0 library. See the end of this topic for further information and acknowledgements.

A complete treatment of the topic of regular expressions could--and does--fill an entire book. *Mastering Regular Expressions*, by Jeffrey Friedl is one such book, and a good one at that. This help topic was written to acquaint the typical user with the most common regular expression features, without getting too bogged down in fine details. The advanced reader is encouraged to seek out additional information on the web, or within the PCRE documentation itself. We have posted one such reference document on our site for your convenience.

Regular Expressions are *very* powerful, and can be more easily understood by studying several examples.

Matching a Single Character
The dot (.) will match any single character, except the newline character. Example: `p.t` will match `pat`, `pet`, `pit`, `pot`, and `put`, and in fact any 3-character sequence with `p` and `t` at its ends and a single character in the middle.

Matching with an Asterisk
The asterisk (`*`) will match zero or more occurrences of the preceding character. Example: `zo*m` will match `zm`, `zom`, `zoom` and `zooooooooom`, among others. Note that the character preceding the asterisk can be the dot, so zero or more occurrences of *any* character will be matched when the construction `.*` is used. Example: `Bo.*r` will match `Boxer`, `Bowler`, `Bookmaker`, `Bookkeeper` and `Building Manager`.

Matching with a Plus Sign
The plus sign (`+`) will match one or more occurrences of the preceding character. Example: `ho+p` will match `hop`, `hoop` and `hooooooop`, among others. Note that the character preceding the plus sign can be the dot, so that one or more occurrences of *any* character will be matched when the construction `.+` is used.

Patterns that use either `*` or `+` can often result in more than one possible matching string. This concept is known as minimal or maximal matching. You can control whether Boxer will return the shortest or longest matching string using the *Maximal matching* checkbox on any dialog where regular expressions are permitted.

Matching at Start of Line
The caret (`^`) can be used to force a match to occur at the start of a line. Example: `^The` will match any line beginning with `The`.

You can also force a start-of-line match using the checkbox provided on the dialog.

Matching at End of Line

The dollar sign ($) can be used to force a match to occur at the end of a line. Example: `result$` will match any line ending with the word `result`.

 You can also force an end-of-line match using the checkbox provided on the dialog.

Character Classes or Range Expressions
One or more characters can be placed within square brackets to designate the characters which can match in that position. Example: `p[aeiou]t` will match `pat`, `pet`, `pit`, `pot` and `put`. Note that digits are also characters, so an expression such as `201[1234]` will match any of `2011`, `2012`, `2013` or `2014`.

Characters can also be placed within square brackets with a dash between them to designate a range of characters. Example: `[b-d]ent` will match `bent`, `cent` and `dent` because the expression `[b-d]` is shorthand for all characters in that range. The character range can be entered in ascending or descending order; both `[A-Z]` and `[Z-A]` are allowed and are functionally equivalent.

The character set appearing within square brackets can be negated by using the caret (`^`) as the first character within the opening square bracket. Example: `[^cb]ent` will match `tent`, `rent`, `sent`, `dent` and others, but *not* `cent` or `bent`. The caret can also be applied to negate a character range within square brackets: `[^a-e]` will match all characters *except* `a`, `b`, `c`, `d` and `e`. If the caret appears anywhere else within the range expression, its meaning reverts to that of matching the caret itself.

Matching Multiple Strings
The vertical rule (`|`) can be used to separate two or more regular expressions so that any of the patterns will match. Example: `red|green|blue|yellow` will match any of the color names that are separated by the vertical rules.

Subpatterns
Left and right parentheses can be used to start and end a *subpattern*. Example: `c(ar|en|oun)t` will match `cart`, `cent` and `count`. In absence of the parentheses, `car|en|ount` would match `car`, `en` or `ount`... a very different result.

Escape Character
The backslash can be used to remove significance from a pattern matching character. Example: if you need to search for an asterisk, use `\*`. To search for a dot, use `\.`. To search for a plus sign, use `\+`. To search for the backslash itself, use `\\`.

 You can also remove significance from pattern matching characters by placing them inside a range expression. For example, `[*+]` could be used to match either an asterisk or a plus sign.

Matching Whole Words
To force a pattern to find only those occurrences of a search string which appear as whole words, the pattern can be surrounded with a sequence that forces a match at a word boundary. Example: to find the word `sign`, but not words such as `assign`, `signature` or `assignment`, use `\bsign\b`.

 You can also force a whole word match using the checkbox provided on the dialog.

Matching Special Characters
Several characters that are not readily typed from the keyboard can be matched using special character sequences:

`\\`	match a backslash character
`\a`	match a bell (alarm) character (ASCII 7)
`\b`	match a backspace character (ASCII 8) (only if used in a character class)
`\cx`	match character Control-x (x = any character)
`\e`	match an escape character (ASCII 27)
`\f`	match a formfeed character (ASCII 12)
`\n`	match a newline character (ASCII 10)
`\r`	match a carriage return character (ASCII 13)
`\t`	match a tab character (ASCII 9)
`\ddd`	match *octal* character ddd (d = any digit 0-7)
`\xhh`	match *hexadecimal* character hh (h = any hex digit)

Generic Character Types
There are several convenient shorthand sequences for matching common character classes:

`\d`	match a decimal digit (0-9), equivalent to: `[0-9]`
`\D`	match any character except a decimal digit, equivalent to: `[^0-9]`
`\s`	match any whitespace character, equivalent to: `[\t\n\f\r ]`
`\S`	match any character except whitespace, equivalent to `[^\t\n\f\r ]`
`\w`	match any word character, equivalent to: `[_a-zA-Z]`
`\W`	match any character except a word character, equivalent to: `[^_a-zA-Z]`

 A word character is considered to be any letter, digit or underscore. No consideration is made for accented characters that reside above value 128 in the character set. If you require such characters in a pattern, you'll need to name these characters explicitly, perhaps in a range expression that also uses `\w`.

Assertions
The following sequences can be used to force a match to occur only at a required position:

\b	match at a word boundary
\B	match when not at a word boundary
\A	match at start of subject
\Z	match at end of subject or before newline
\z	match at end of subject
\G	match at first matching position in subject

Useful Constructions
The following examples illustrate some common constructions, and give examples of the utility--and complexity--of some advanced regular expressions:

.*	match zero or more occurrences of any character
.+	match one or more occurrences of any character
^$	match an empty line
^\s+$	match a line containing only *whitespace*
^\s+	match leading whitespace
\s+$	match trailing whitespace
[a-zA-Z]	match any alphabetic character
this|that	match 'this' or 'that'
\b(\w+)\s+\1\b	match repeated words (such as 'the the')
\b[A-Z0-9._%-]+@[A-Z0-9._%-]+\.[A-Z]{2,4}\b	match a valid email address

Min/Max Quantifiers
A min/max quantifier can be used to control how many instances of the preceding entity are to be allowed within a match. The syntax for min/max quantifiers is summarized in this table:

{	start a min/max quantifier
}	end a min/max quantifier
{3}	match exactly 3 of the previous item
{3,}	match at least 3 of the previous item
{3,5}	match at least 3, but no more than 5 of the previous item

Example: the pattern `[abc]{4,8}` would match a sequence of characters consisting of the letters a, b or c, so long as at least 4 characters are present, and no more than 8 appear. Potential matches: `aaaa`, `accb`, `abcabc`, `bbbbcccc`. Non matches: `aaa`, `abcd`, `abcabcabc`.

☞ Careful readers might observe that `*` is effectively shorthand for `{0,}` and `+` is shorthand for `{1,}`.

Back References and Named Subpatterns
One of the more powerful features of Perl regular expressions is the ability to make reference within a pattern to the string that matched a subpattern which occurred earlier in the pattern. Subpatterns are created when a portion of a pattern is enclosed in left and right parentheses. The first opening left parenthesis encountered starts a subpattern whose number is 1. The second left parenthesis creates subpattern 2, and so on. To make a back reference to a subpattern by number, this syntax is used:

`\1`	back reference to subpattern number 1

Referring to subpatterns by number can get confusing when a complex regular expression is being created. For this reason, *named subpatterns* are also permitted. To start a subpattern named 'foo', the following syntax would be used:

`(?P<foo>`	start a subpattern named 'foo'

Later on in the pattern, the string that matched subpattern 'foo' could be referenced using this syntax:

`(?P=foo)`	back reference to the subpattern named 'foo'

The example presented above that matches repeated words used a back reference:

```
\b(\w+)\s+\1\b
```

The subpattern `(\w+)` matches any string that contains one or more word characters. In order for the entire pattern to match, that same string must appear again (due to the `\1` reference) with one or more spaces (`\s+`) in between. Finally, the `\b` sequences at each end ensure that the pattern matches only at a word boundary.

☞ Named subpattern references can also be used in the replace string of the Replace and Replace Line Enders commands, and with the ChangeStringRE() macro function.

Closing Example
Finally, it's worth mentioning that any or all of the expressions presented above can be used within the same regular expression. This artificially complex example:

```
^The\sq[^a]ic{1}k.*f[aeiou]x.*ov[a-e]r.*lazy\040dog\.$
```

would match the sentence:

```
The quick brown fox jumped over the lazy dog.
```

so long as it appeared on a single line.

PCRE 5.0 License

The Perl-Compatible Regular Expression (PCRE) package used by Boxer was written by Philip Hazel, and is used in accordance with the PCRE license:

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of the University of Cambridge nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

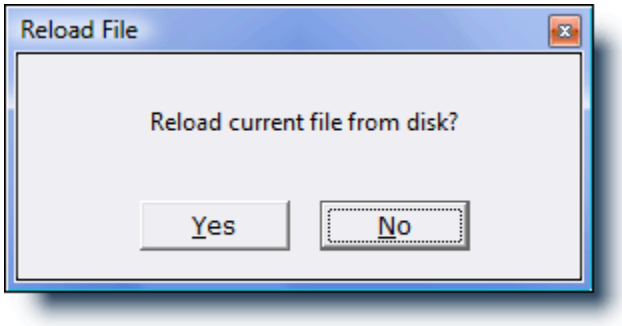
5.218 Reload

Menu: File > Reload

Default Shortcut Key: Shift+Ctrl+O

Macro function: ReloadFile()

The Reload command can be used to reload the current file from disk, thereby losing any unsaved changes in the file. Because of the potential for losing changes accidentally, a dialog box will appear to confirm your intention to reload.



The most common use for this command is to restart the editing of a file after having made a large number of unwanted changes. The Undo command can be used to step back through the changes made, but its limits can be exhausted if a large number of changes are made. In that case, the Reload command must be used.

5.219 Remove

Menu: Project > Remove

Default Shortcut Key: none

Macro function: ProjectRemove()

Use the Remove command to remove the current file from the active project.

☞ See the Project | New command for full details about Boxer's project file feature.

5.220 Repeat Last Command

Menu: Tools > Repeat Last Command

Default Shortcut Key: F10

Macro function: none

Repeat Last Command can be used to repeat the most recently issued command. This command is especially convenient when the command last issued does not have a shortcut key, and must be executed from the pull-down menus. Repeat Last Command is assigned to the *F10* key, by default, to ensure it can be easily executed.

☞ Repeat Last Command cannot be used to repeat cursor movement commands such as Up, Down, Left, Right, etc.

💡 Repeat Last Command can also be used to repeat the insertion of single characters. When a character is not readily typed from the keyboard (think high-ASCII characters), Repeat Last Command can be a real time saver.

5.221 Replace

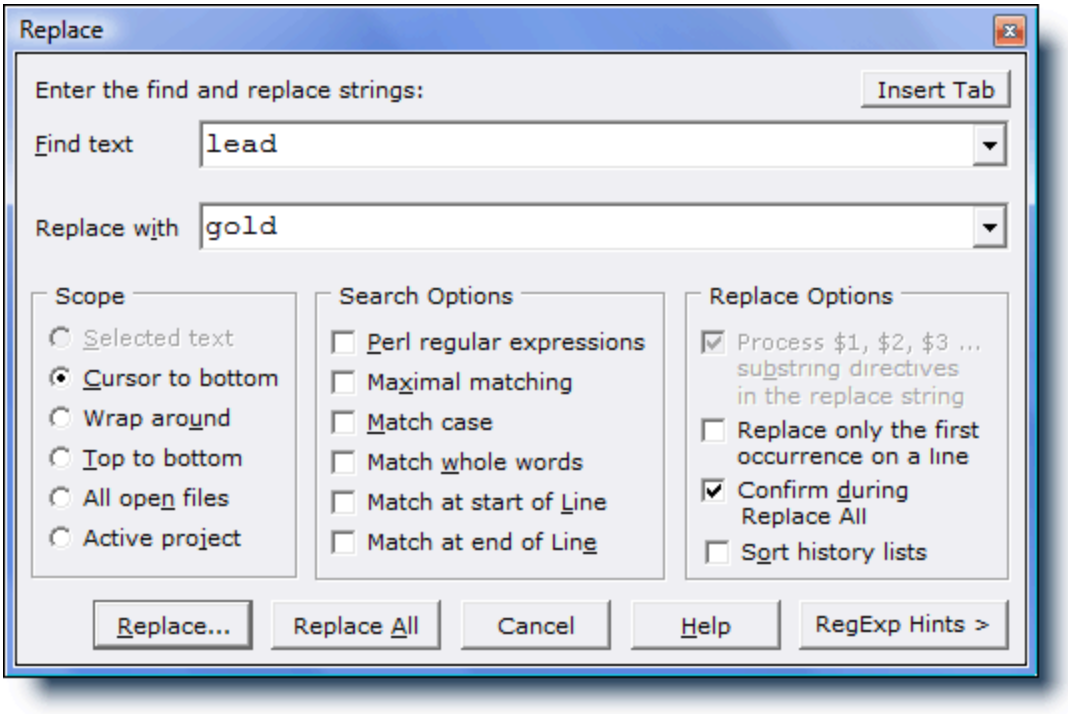
Menu: Search > Replace

Default Shortcut Key: Ctrl+R

Macro function: Replace()

The Replace command can be used to search for a text string and replace it with another string. Replacements can be made selectively or globally, within the current file or across all edited files. Regular Expressions can be entered within the search string.

The controls and options in the Replace dialog box are described below:



Find text
This is the edit box where the search string is entered. When the Replace command is issued, the word beneath the text cursor is placed into the *Find Text* edit box, in case that word--or a word which is nearly the same--is to be the search string. To recall a search string which was previously entered, use the drop-down list or press the up or down arrow keys to review the items in the history list. Regular Expressions may be used within the search string.

The *Delete* key can be used while the drop-down list is displayed to delete a selected entry from the history list.

Special characters can be entered into the *Find text* edit box using the technique

described in the Help topic Inserting Special Characters.

Replace with
This is the edit box where the replace string is entered. To recall a replace string which was previously entered, use the drop-down list or press the up or down arrow keys to review the items in the history list.


The *Delete* key can be used while the drop-down list is displayed to delete a selected entry from the history list.


The Replace command is line-oriented. It considers each line individually and does not look across line enders to match a search string which might span lines. Consequently, it is not possible to create new line enders using the Replace command, nor to delete existing line enders. For these types of operations, the Replace Line Enders command must be used.

Insert Tab
Use this button to insert a tab character into the *Find Text* or *Replace with* edit boxes.

Ordinarily, the *Tab* key is used to move from field to field within a dialog box. If you would prefer that the Tab key insert a tab character in this dialog box, and in other Find/Replace related dialog boxes, check the relevant box on the Configure | Preferences | Tabs dialog page.

Scope

Selected text
This option can be used to restrict the search and replace operation to the extent of the selected text.

Cursor to bottom
This option causes the search and replace operation to be performed from the cursor onward, toward the end of file. (There is no provision for making replacements in a backward direction.)

Wraparound
This option causes the search to be performed from the cursor onward, toward the end of file. When the end of file is reached, the search resumes at the top and continues to the original cursor position.

Top to bottom
This option causes the search and replace operation to be performed from the top of file onward, toward the end of file. (There is no provision for making replacements in a backward direction.)

All open files
This option causes the search and replace operation to be performed across all open files.

Active project
Use this option to limit the scope of the Replace operation to those files within the

active project.

Search Options

Perl regular expressions
If this box is checked, wildcard characters within the search string will be interpreted according to the Perl-Compatible Regular Expression (PCRE) convention. In part, this means that the asterisk (`*`) will cause a match of zero or more occurrences of the preceding character. The period (`.`) will match any single character. For more information, see Regular Expressions.

Maximal matching
When using pattern matching characters, there can sometimes be more than one text string that matches the search string. This option can be used to request that the longest possible matching string be returned.

Match case
This option can be used to force the search string to be matched exactly. When unchecked, a case insensitive search is performed.

Match whole words
This option can be used to restrict matches to those strings which appear as a whole word. The characters which serve to delimit words are user-configurable; see Configure | Preferences | Cursor.

Match at start of line
This option can be used to force the search string to be matched only when a matching string appears at the start of a line. This effect can also be achieved with a Regular Expression.

Match at end of line
This option can be used to force the search string to be matched only when a matching string appears at the end of a line. This effect can also be achieved with a Regular Expression.

Replace Options

Process $1, $2, $3... substring directives in the replace string
When this option is checked, special directives in the replace string will be replaced at match-time with subpatterns from the search string. This is a very powerful feature, as the following examples will illustrate.

Example 1:

Find text: `(\w+),(\w+)`
Replace with: `$2 $1`

The search string will match a string of one or more word characters followed by a comma, followed by another string of one or more word characters. For example: `Smith,John`. The parentheses are used to define subpatterns. The first open parenthesis indicates subpattern number 1, the next number 2, and so on. In this way,

the replace string can vary depending on what the search string matches. If the string `Smith,John` is matched, then the replace string will be `John Smith`. Running this search and replace operation on a data file would have the effect of inverting a list of `Lastname,Firstname` data to `Firstname Lastname` format.

Example 2:

Find text: `(Boxer|BOXER)`
Replace with: `<b>$1</b>`

This search string will match either `Boxer` or `BOXER`. The replace string will be equal to whatever the string matched, surrounded by the HTML open-bold and close-bold sequences. In this way, the target word can be replaced without regard to its case, while ensuring that no case conversion occurs due to the replacement.

☞ The entire matching string is designated as `$0`, even if subpatterns are not used. Up to 100 subpatterns can be referenced, numbering from `$0` to `$99`.

Example 3:

Find text: `"([^"]+),([^"]*)"`
Replace with: `"$1$2"`

This pair of search and replace strings can be used to remove commas from within the data fields of quote and comma-delimited data, without disturbing the commas that are used as field separators. The search pattern matches an entire double-quoted data field, so long as a comma appears within the data with at least one character to its left. The replace string references the data on either side of the comma as `$1` and `$2`, resulting in a replace string that duplicates the string matched, except that the comma is excluded. (If multiple commas appear within a single data field, you'll need to run the replace operation repeatedly until all occurrences have been replaced.)

☞ References to *named subpatterns* such as `(?P=name)` are also recognized in the replace string. See the Regular Expressions topic for more information about named subpatterns.

Replace only the first occurrence on a line
When this option is checked, only the first matching instance on a line will be considered eligible for replacement.

Confirm during Replace All
When this option is selected the *Replace All* operation will prompt before making each replacement. A dialog box will be presented so that each replacement can be confirmed. From this confirmation dialog box it is possible to later opt for unconditional replacements, by selecting its *All* button.

Sort history lists
If this box is checked the search and replace history lists will be maintained in alphabetic order, rather than in the order the strings were entered.

When switching to alphabetically sorted lists, the chronological ordering of the lists will be lost, and cannot be restored by unchecking the checkbox.

No attempt is made to associate the history list entries with the time that they were added to the list. If a sorted history list is used consistently, over time the list will come to hold an unrepresentative set of search phrases. In the extreme case, after many Replace operations, a list could result that contained only phrases beginning with the letter 'A'. This occurs because entries at the bottom of the list will be removed after the maximum size of the list is reached.

5.222 Replace Again

Menu: Search > Replace Again

Default Shortcut Key: Shift+Ctrl+R

Macro function: ReplaceAgain()

The Replace Again command is used to repeat the most recent search and replace operation. The new operation will obey all of the options which were used when the previous search and replace operation was initiated with the Replace command.

5.223 Replace Line Enders

Menu: Search > Replace Line Enders

Default Shortcut Key: Ctrl+Alt+R

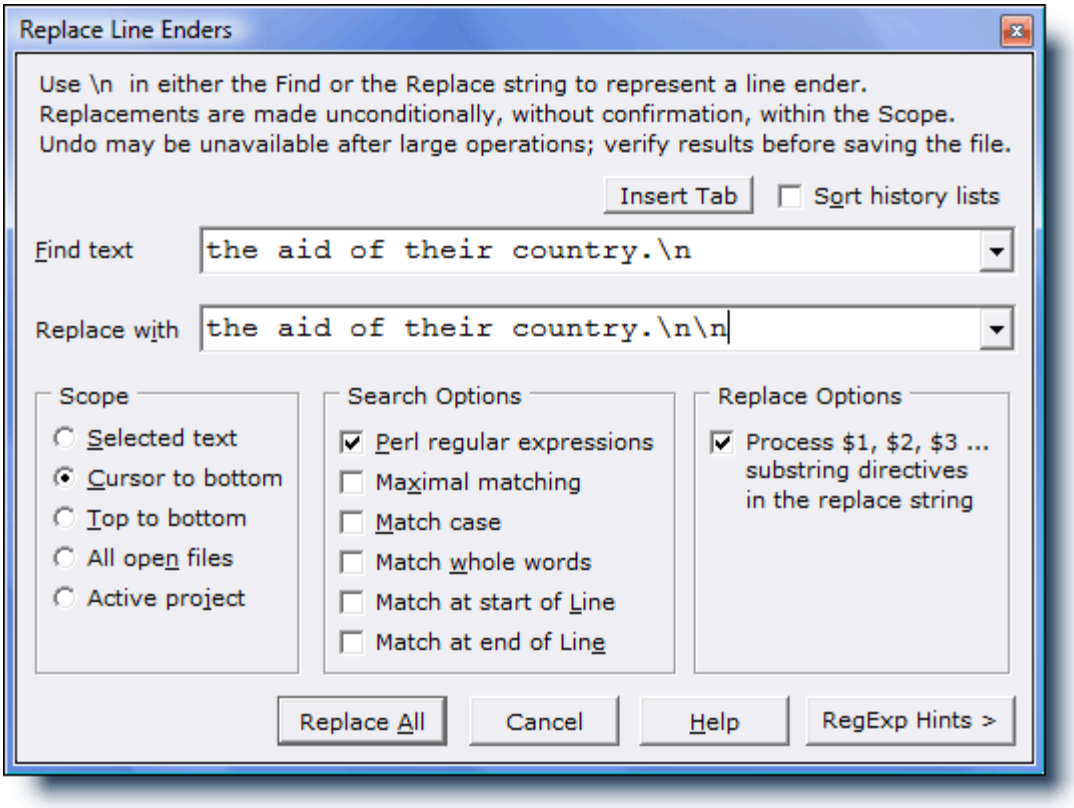
Macro function: ReplaceLineEnders()

The Replace Line Enders command can be used to search for a text string and replace it with another string. Replacements can be made selectively or globally, within the current file or across all edited files. Regular Expressions can be entered within the search string.

Unlike the Replace command, the Replace Line Enders command can be used to perform search and replace operations that span lines, and which may result in the addition or removal of lines from the file.

Important Note: The Replace Line Enders command performs its replacements **unconditionally, without user confirmation.** When large operations are performed, Undo may be unavailable. For these reasons, it's advisable to **make a backup copy** of your file before using this command.

The controls and options in the Replace Line Enders dialog box are described below:



Find text
This is the edit box where the search string is entered. When the Replace Line Enders command is issued, the word beneath the text cursor is placed into the *Find Text* edit box, in case that word--or a word which is nearly the same--is to be the search string. To recall a search string which was previously entered, use the drop-down list or press the up or down arrow keys to review the items in the history list. Regular Expressions may be used within the search string.

Use the sequence `\n` to represent a line ender (newline). Example: if you are searching for a line that ends with 'jelly' that occurs just before a line that starts with 'bean', your search string would be: `jelly\nbean`

The *Delete* key can be used while the drop-down list is displayed to delete a selected entry from the history list.

Special characters can be entered into the *Find text* edit box using the technique described in the Help topic Inserting Special Characters.

Replace with
This is the edit box where the replace string is entered. To recall a replace string which was previously entered, use the drop-down list or press the up or down arrow keys to review the items in the history list.

Use the sequence `\n` to represent a line ender (newline). Example: if you wanted to

add two blank lines after all lines that end with the text `THE FOLLOWING:`, theses search and replace strings would be used:

Find text: `THE FOLLOWING:\n`
Replace with: `THE FOLLOWING:\n\n\n`


The *Delete* key can be used while the drop-down list is displayed to delete a selected entry from the history list.

Insert Tab
Use this button to insert a tab character into the *Find text* or *Replace with* edit boxes.

Ordinarily, the *Tab* key is used to move from field to field within a dialog box. If you would prefer that the Tab key insert a tab character in this dialog box, and in other Find/Replace related dialog boxes, check the relevant box on the Configure | Preferences | Tabs dialog page.

Scope

Selected text
This option can be used to restrict the search and replace operation to the extent of the selected text.

Cursor to bottom
This option causes the search and replace operation to be performed from the cursor onward, toward the end of file. (There is no provision for making replacements in a backward direction.)

Top to bottom
This option causes the search and replace operation to be performed from the top of file onward, toward the end of file. (There is no provision for making replacements in a backward direction.)

All open files
This option causes the search and replace operation to be performed across all open files.

Active project
Use this option to limit the scope of the Find operation to those files within the active project.

Search Options

Perl regular expressions
If this box is checked, wildcard characters within the search string will be interpreted according to the Perl-Compatible Regular Expression (PCRE) convention. In part, this means that the asterisk (`*`) will cause a match of zero or more occurrences of the preceding character. The period (`.`) will match any single character. For more information, see Regular Expressions.

Maximal matching
When using pattern matching characters, there can sometimes be more than one text string that matches the search string. This option can be used to request that the longest possible matching string be returned.

Match case
This option can be used to force the search string to be matched exactly. When unchecked, a case insensitive search is performed.

Match whole words
This option can be used to restrict matches to those strings which appear as a whole word. The characters which serve to delimit words are user-configurable; see Configure | Preferences | Cursor.

Match at start of line
This option can be used to force the search string to be matched only when a matching string appears at the start of a line. This effect can also be achieved with a Regular Expression.

Match at end of line
This option can be used to force the search string to be matched only when a matching string appears at the end of a line. This effect can also be achieved with a Regular Expression.

Replace Options

Process $1, $2, $3... substring directives in the replace string
When this option is checked, special directives in the replace string will be replaced at match-time with subpatterns from the search string. This is a very powerful feature, as the following examples will illustrate.

Example 1:

Find text: `(\w+),(\w+)`
Replace with: `$2 $1`

The search string will match a string of one or more word characters followed by a comma, followed by another string of one or more word characters. For example: `Smith,John`. The parentheses are used to define subpatterns. The first open parenthesis indicates subpattern number 1, the next number 2, and so on. In this way, the replace string can vary depending on what the search string matches. If the string `Smith,John` is matched, then the replace string will be `John Smith`. Running this search and replace operation on a data file would have the effect of inverting a list of `Lastname,Firstname` data to `Firstname Lastname`.

Example 2:

Find text: `(Boxer|BOXER)`
Replace with: `<b>$1</b>`

The search string will match either `Boxer` or `BOXER`. The replace string will be equal to

whatever the string matched, surrounded by the HTML open-bold and close-bold sequences. In this way, the target word can be replaced without regard to its case, while ensuring that no case conversion occurs due to the replacement.

☞ The entire matching string is designated as $0, even if subpatterns are not used. Up to 100 subpatterns can be referenced, numbering from $0 to $99.

Sort history lists
If this box is checked the search and replace history lists will be maintained in alphabetic order, rather than in the order the strings were entered.

☞ When switching to alphabetically sorted lists, the chronological ordering of the lists will be lost, and cannot be restored by unchecking the checkbox.

☞ No attempt is made to associate the history list entries with the time that they were added to the list. If a sorted history list is used consistently, over time the list will come to hold an unrepresentative set of search phrases. In the extreme case, after many Replace operations, a list could result that contained only phrases beginning with the letter 'A'. This occurs because entries at the bottom of the list will be removed after the maximum size of the list is reached.

5.224 Restore All

Menu: Window > Restore All

Default Shortcut Key: none

Macro function: RestoreAll()

The Restore All command can be used to return all minimized windows to their former sizes and positions.

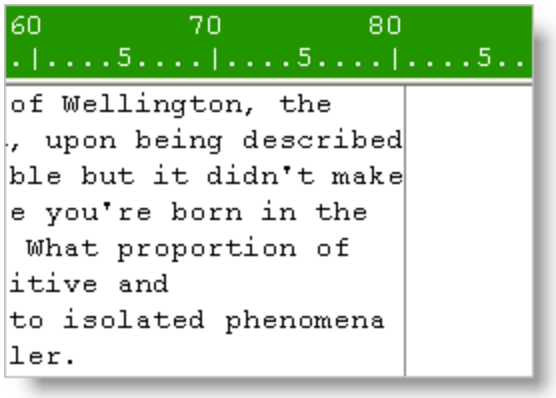
5.225 Right Margin Rule

Menu: View > Right Margin Rule

Default Shortcut Key: Alt+F6

Macro function: ViewRightMarginRule()

The View Right Margin Rule command is used to toggle on and off the display of a thin vertical line which marks a user-defined column. The Right Margin Rule can be used as a visual reminder that a particular line length has been exceeded.



You can set the Right Margin Rule to any column you desire; the default is column 80. An option is provided to set the column on the Configure | Preferences | Display options page. The option is titled *Show right margin rule at column...*

Clicking on the Right Margin Rule with the right mouse button provides access to its context menu, which allows the line to be turned off.

5.226 Right Window Edge

Menu: Jump > Right Window Edge

Default Shortcut Key: none

Macro function: RightWindowEdge()

The Right Window Edge command positions the text cursor to the right edge of the current window.

If the cursor has been constrained to move from the end of a line to the start of the next line, the behavior of the Right Window Edge command will be impacted. In this case, when issued on a line that does not reach the right window edge, this command will move to the cursor to the end of the line.

If a text selection is present when this command is issued, the selection will be extended to the new cursor location.

5.227 ROT5

Menu: Block > Convert Other > ROT5

Default Shortcut Key: none

Macro function: ROT5()

This command will apply a ROT5 (rotation 5) conversion to the selected text. If an entire file is to be converted, use the Select All Text command to select the whole file.

ROT5 is a simple substitution cypher that replaces each digit (0-9) with the digit that resides 5 positions higher in sequence. The digit '0' would be replaced by '5', '1' by '6', and so on. When adding 5 would exceed the digit '9', the conversion wraps back to the beginning of the sequence.

ROT5 is sometimes used to disguise numeric text from casual viewing, such as when the answer to a puzzle is presented alongside the question. Applying the conversion to ROT5-encoded text a second time returns the original text. Additional information about ROT13 and other cyphers can be found on Wikipedia.

5.228 ROT13

Menu: Block > Convert Other > ROT13

Default Shortcut Key: none

Macro function: ROT13()

This command will apply a ROT13 (rotation 13) conversion to the selected text. If an entire file is to be converted, use the Select All Text command to select the whole file.

ROT13 is a simple text substitution cypher that replaces each alphabetic character (A-Z and a-z) with the character that resides 13 positions forward in the alphabet. The letter 'A' would be replaced by 'N', 'B' by 'O', and so on. When adding 13 would exceed the letter 'Z', the conversion wraps back to the beginning of the alphabet.

ROT13 is sometimes used to disguise text from casual viewing, such as when the answer to a puzzle is presented alongside the question. Some entries in the *Windows Registry* are ROT13 encoded. Applying the conversion to ROT13-encoded text a second time returns the original, human-readable text. Additional information about ROT13 can be found on Wikipedia.

5.229 ROT18

Menu: Block > Convert Other > ROT18

Default Shortcut Key: none

Macro function: ROT18()

This command will apply a ROT18 (rotation 18) conversion to the selected text. If an entire file is to be converted, use the Select All Text command to select the whole file.

ROT18 is a simple text substitution cypher that replaces each alphabetic character (A-Z and a-z) with the character that resides 13 positions forward in the alphabet, and each digit (0-9) with the digit that resides 5 positions higher in sequence. A ROT18 conversion is thus equivalent to applying the ROT5 and ROT13 conversion commands to the same text.

ROT18 is sometimes used to disguise text from casual viewing, such as when the answer to a puzzle is presented alongside the question. Applying the conversion to ROT18-encoded text a second time returns the original, human-readable text.

Additional information about ROT13 and other cyphers can be found on Wikipedia.

5.230 ROT47

Menu: Block > Convert Other > ROT47

Default Shortcut Key: none

Macro function: ROT47()

This command will apply a ROT47 (rotation 47) conversion to the selected text. If an entire file is to be converted, use the Select All Text command to select the whole file.

ROT47 is a simple text substitution cypher that replaces each character value in the ASCII range 33 to 126 inclusive with the character whose value is 47 positions higher. When adding 47 would exceed character value 126, the conversion wraps back to the beginning of the sequence.

ROT47 is sometimes used to disguise text from casual viewing, such as when the answer to a puzzle is presented alongside the question. Applying the conversion to ROT47-encoded text a second time returns the original, human-readable text. Additional information about ROT13 and other cyphers can be found on Wikipedia.

5.231 Save

Menu: File > Save

Default Shortcut Key: Ctrl+S

Macro function: Save()

The Save command is used to write the contents of the current file to disk. If the file being edited is a new file with a temporary untitled name, the Save As command will be performed automatically so that a name can be provided. The Save command will be disabled when there are no changes in the current file that need to be saved.


If the Save command is issued while text is selected, a dialog box can appear to get the name of the file to which the selected text should be saved. This option is off by default, but can be enabled on the Configure | Preferences | File I/O options page. The option is titled *File Save performs Save Selection As, when text is selected*. This option page also contains other configuration options which relate to saving files.

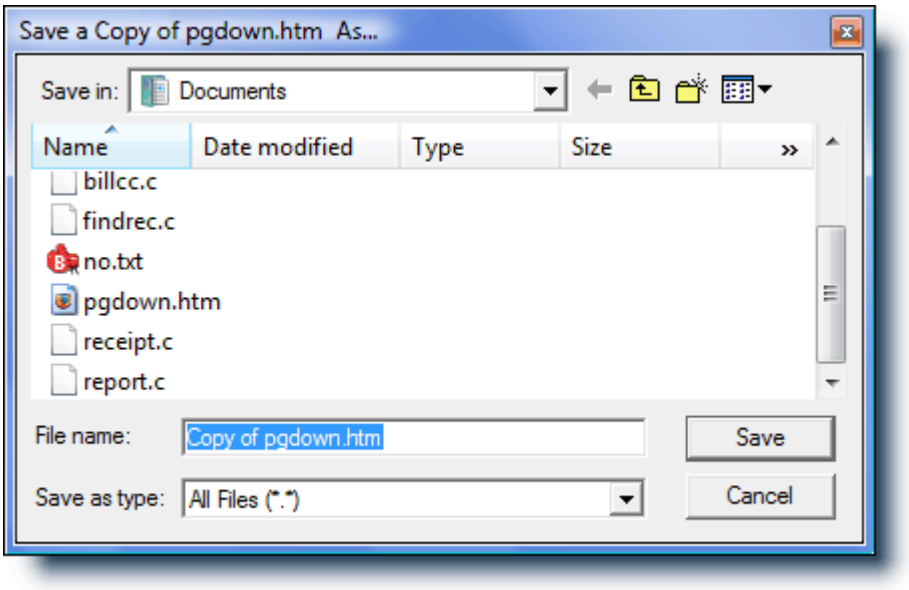
5.232 Save a Copy As

Menu: File > Save a Copy As

Default Shortcut Key: none

Macro function: SaveACopyAs()

The Save a Copy As command is used to save the current file to disk under a new filename. A standard File Save dialog box will appear so that a new name can be specified. Unlike the Save As command, the editor does not record the new filename for use by future save operations. This command can be useful for creating progressive copies of a file during a long edit session so that a change history is created, or at anytime when a copy might be useful.



Boxer will not automatically add a file extension to the filename you provide; you should add the desired file extension yourself.

If the Save a Copy As command is used to save a file which is being viewed in read-only Hex Mode, the hex view of the file--and not the file's true content--is what will be saved to disk. If a file was opened for editing in hex mode, then Save a Copy As will create a copy of the file's actual content.

5.233 Save All

Menu: File > Save All

Default Shortcut Key: Shift+Ctrl+S

Macro function: SaveAll()

The Save All command can be used to write the contents of all files with unsaved changes to disk. If any of the files being edited are new files with a temporary untitled name, the Save As command will be performed automatically so that a name can be provided. The Save All command will be disabled when there are no changes in any open files that need to be saved.

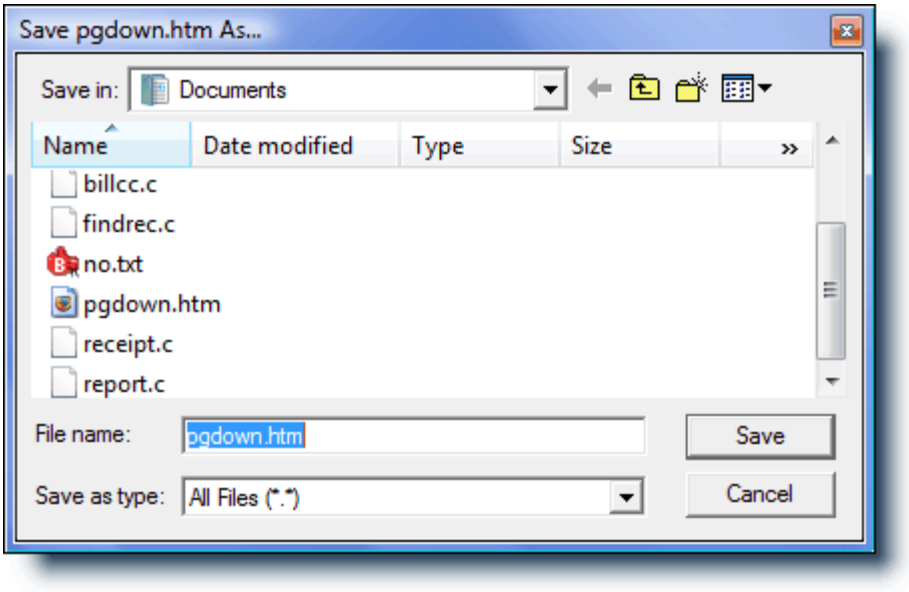
5.234 Save As

Menu: File > Save As

Default Shortcut Key: F12

Macro function: SaveAs()

The Save As command is used to save the current file to disk using a new filename. A standard Windows File Save dialog box will appear so that a new name can be specified. The file will remain on disk under its old name (except when an untitled file is being saved), and a copy of the file will be saved to the new name provided. Boxer will then record the file's new name so that all future save operations are made to the new name.



If you would like to change either the line ender style (PC, Macintosh, or Unix), or the file encoding (ASCII, UTF-8, UTF-16), visit the File Properties dialog before using the Save As command to save the file.

Boxer will not automatically add a file extension to the filename you provide; you should add the desired file extension yourself.

If the Save As command is used to save a file which is being viewed in read-only Hex Mode, the hex view of the file--and not the file's true content--is what will be saved to disk. If a file was opened for editing in hex mode, then Save As will create a copy of the file's actual content.

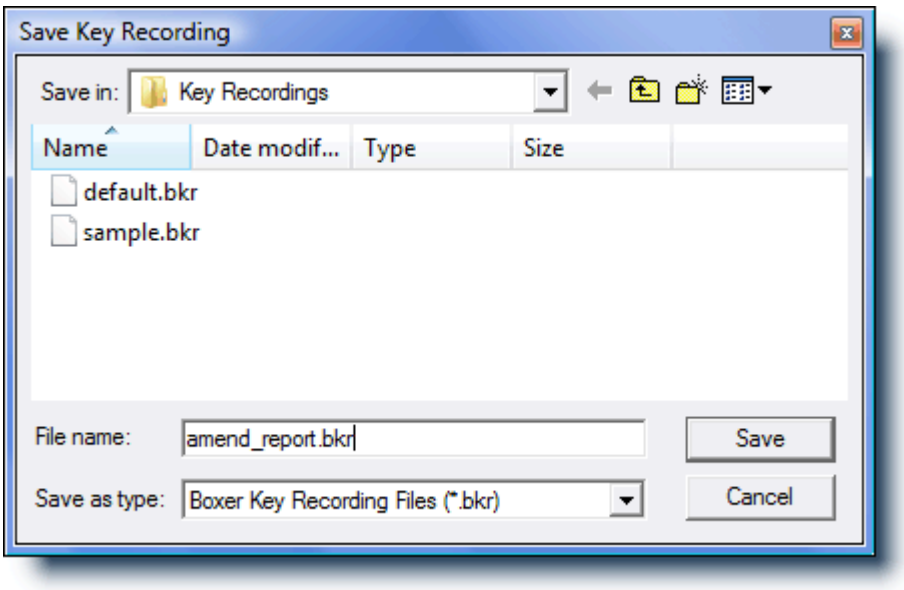
5.235 Save Key Recording

Menu: Tools > Save Key Recording

Default Shortcut Key: none

Macro function: none

Use the Save Key Recording command to save a key recording to a disk file. A dialog will appear that allows a name to be entered. By default, key recordings are stored in Boxer's 'Key Recordings' subdirectory.



Command key assignments are not used in the files created by Save Key Recording, so there's no chance that changing key assignments might disturb an existing key recording. Likewise, key recording files can be freely shared with other Boxer users without concern that playback might be influenced by differing key assignments on the target machine. However, depending on the nature of the key recording, default settings on the target machine (autoindent, typing wrap, etc) could influence how a key recording performs upon playback.

5.236 Save Selection As

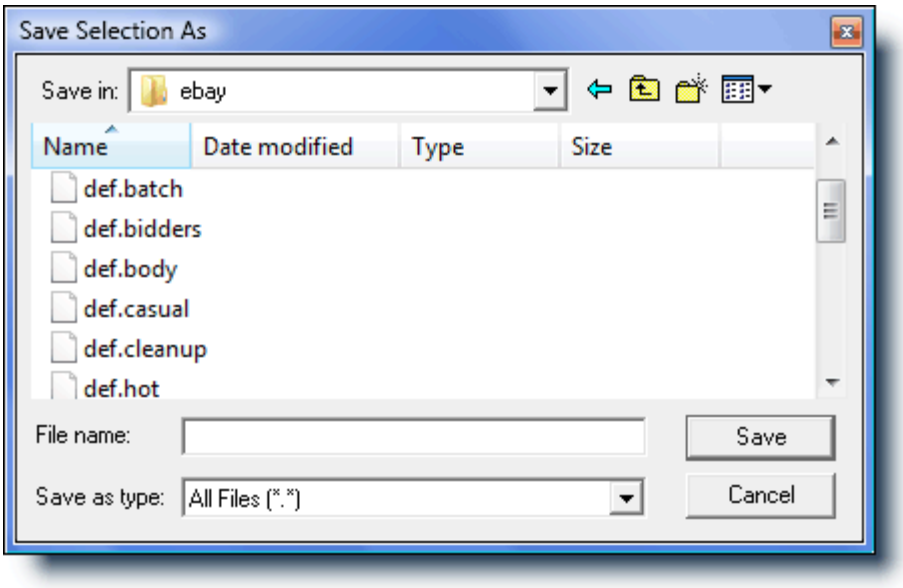
Menu: Block > Save Selection As

Default Shortcut Key: none

Macro function: SaveSelectionAs()

The Save Selection As command brings up the standard Windows Save dialog and

prompts the user for a filename under which to save the selected text.



☞ Boxer will not automatically add a file extension to the filename you provide; you should add the desired file extension yourself.

☞ The current file encoding and line ender style, as indicated in File Properties, will be used for the file that's created. If you want to save the selected text to a file using different encoding or line enders, paste the text into a new file and set the properties of the new file before saving.

☞ If the Save command is issued while text is selected, a dialog box can appear to get the name of the file to which the selected text should be saved. This option is off by default, but can be enabled on the Configure | Preferences | File I/O options page. The option is titled *File Save performs Save Selection As when text is selected*. This option page also contains other configuration options which relate to loading and saving files.

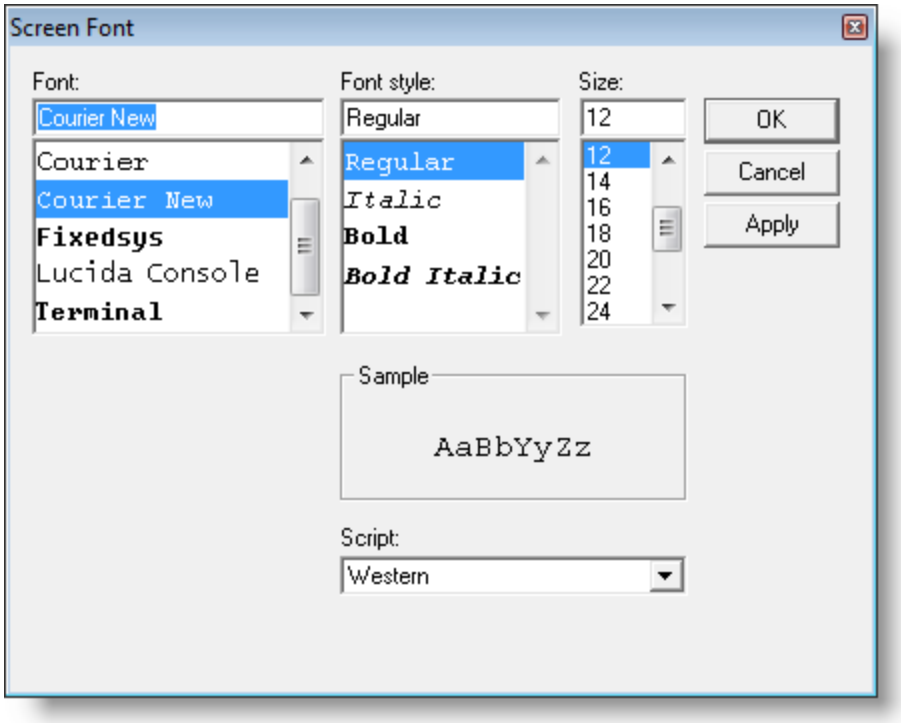
5.237 Screen Font

Menu: Configure > Screen Font

Default Shortcut Key: none

Macro function: ConfigureScreenFont()

The Screen Font command is used to select the font that is used to display edited files on-screen. The standard Windows font dialog is presented for selecting the screen font:



Boxer requires that *fixed width* fonts be used, so the Screen Font dialog box does not display *proportionally spaced* fonts. This is required, in part, to ensure that columnar selections can be highlighted neatly in rectangular blocks, and so that the Column Ruler can be used. These features would not be possible if the use of proportional fonts were permitted.

The *Font* listbox at the left of the dialog displays the *fixed width* fonts which are available for selection.

The *Font Style* listbox display the styles which are available for the selected font. Typically these are *Regular*, *Italic*, *Bold* and *Bold Italic*, although some fonts may not offer all styles.

The *Size* listbox displays the sizes which are available for the selected font.

The *Script* drop-down list displays the various character mappings which are available for the selected font.

The color that will be used to display the font selected is controlled via the Configure Colors command.

Tips and Notes

You may have need to display files which were created using a DOS program and which contain characters from the upper half of the ASCII character set. To display these files properly select a font which offers an *OEM/DOS* script style. One such font that is available on most systems is the *Terminal* font.

At the time of this writing, the Internet site http://keithdevens.com/wiki/ProgrammerFonts contained good information about fixed width fonts, along with links to several screen fonts which could be downloaded free of charge. The Dina Programming Font is a very nice, free, fixed width font.

Changing the *Font Style* selection from this dialog will cause the font style of 'Normal' text to be changed to the selected style. It will not alter the font styles of other screen syntax elements such as Comments, Reserved Words, Strings, etc. The Configure Colors command can be used to select the font style(s) for these elements, as well as for Normal text. For maximum flexibility it may be advisable to let the font style remain as 'Regular' in the Screen Font dialog and select the font styles to be used for Color Syntax Highlighting from the Configure | Colors dialog.

The Screen Font is not used to print files; see the Printer Font command to select a font for that purpose.

When selecting a True Type font, the standard Windows font dialog box may display a message at the bottom indicating that the selected font will be used for both screen display and printing. This message does *not* apply to the use of fonts within Boxer, since Boxer permits the Screen Font and Printer Font to be selected separately. The message intends to convey the idea that the font is *capable* of being used for both the screen and the printer.

5.238 Scroll Down

Menu: View > Scroll Down

Default Shortcut Key: Ctrl+Up Arrow

Macro function: ScrollDown()

The Scroll Down command will scroll the current file down regardless of where the text cursor is positioned within the window. The cursor remains on the current line until a window edge requires it to be changed. This command is useful to scroll a file down without losing your position in the file. There must be more than a screen full of lines in order for this command to operate.

5.239 Scroll Left

Menu: View > Scroll Left

Default Shortcut Key: Alt+Left Arrow

Macro function: ScrollLeft()

The Scroll Left command will scroll the current file left regardless of where the text cursor is positioned within the window. If column 1 is already visible on screen, no

movement is possible. This command is useful to scroll a file leftward without losing your position in the file.

5.240 Scroll Right

Menu: View > Scroll Right

Default Shortcut Key: Alt+Right Arrow

Macro function: ScrollRight()

The Scroll Right command will scroll the current file right regardless of where the text cursor is positioned within the window. This command is useful to scroll a file rightward without losing your position in the file.

5.241 Scroll Up

Menu: View > Scroll Up

Default Shortcut Key: Ctrl+Down Arrow

Macro function: ScrollUp()

The Scroll Up command will scroll the current file up regardless of where the text cursor is positioned within the window. The cursor remains on the current line until a window edge requires it to be changed. This command is useful to scroll a file up without losing your position in the file. There must be more than a screen full of lines in order for this command to operate.

5.242 Select All Text

Menu: Edit > Select All Text

Default Shortcut Key: Ctrl+A

Macro function: SelectAllText()

The Select All Text command can be used to quickly select all text within the current file. The selected text can then be operated upon in all the ways in which manually selected text might be manipulated.

If the default selection mode is columnar (Edit | Select Columnar), the selection mode must be changed to stream mode (Edit | Select Stream) in order to perform the Select All Text command.

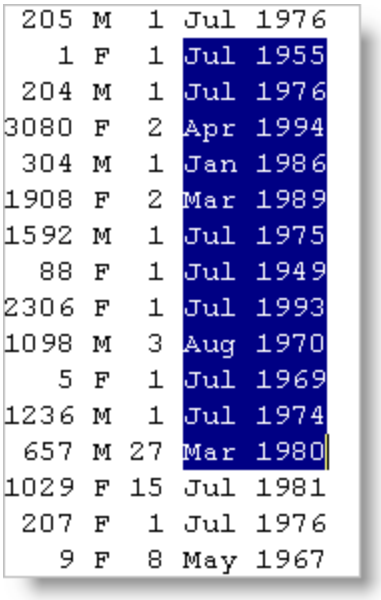
5.243 Select Columnar

Menu: Block > Select Columnar

Default Shortcut Key: Alt+2

Macro function: SelectColumnar()

Columnar Selection
The Select Columnar command is used to set Boxer's *default selection mode* to Columnar. In this mode text is selected in rectangular blocks, as opposed to lines.



The *default selection mode* determines what selection style is used when text selection is initiated either from the keyboard or with the left mouse button. If the *default selection mode* is Stream, a temporary override to Columnar can be achieved by holding down the *Ctrl* key before selecting with the mouse. If a three-button mouse is installed, the center button can be used to perform Columnar text selection regardless of the current selection mode.

Selecting Text with the Keyboard
Text selection can be performed by keyboard by pressing and holding down the *Shift* key and moving the text cursor. Any cursor movement key can be used to move the cursor, e.g., PgUp, PgDn, Home, End, etc., as long as the *Shift* key remains depressed. As the text cursor is moved through the file, the selected area is extended.

Selecting Text with the Mouse
Text selection can be accomplished with the mouse by pressing and holding the left (or center) mouse button and dragging the mouse to highlight the desired text. As the mouse pointer is moved through the file, the selected rectangle is extended. An existing selection can be extended by depressing *Shift* or *Ctrl* before clicking or dragging the mouse.

Selecting the Entire File
The entire file can be selected by issuing the Select All Text command. Doing so while in Columnar mode will result in the default selection mode being changed to Columnar.

Ending Text Selection
Text selection ends when the mouse button or the *Shift* key is released. The selected text remains highlighted and is ready to be operated upon by any of Boxer's Block commands, such as Cut, Copy, Append, Cut-Append, etc.

Extending a Selection with Other Commands
When selected text is present, the Go to Line and Find commands provide options to extend the selection to the new cursor position that results from their operation. The Replace command provides an option to limit the replacement operation to the selected text.

Saving and Printing Text Selections
When text is selected, issuing either the Save or Save Selection As command prompts the user to save the selected area to a disk file. Similarly, when text is selected and the Print command is issued, an option is available to print only the selected area.

Deselecting Text
Selected text can be deselected by clicking once with the left mouse button, pressing the Escape key, or pressing any of the cursor arrow keys. Pressing *Enter*, or any other alphanumeric key will cause selected text to be deleted.


If a columnar selection is started in the virtual space beyond the end of a line, or within the virtual space of a preceding Tab character, the starting column of the selection will be moved to the nearest column to the left which contains a character.


If you start a selection and find that the mode was set to Stream, simply issue this command to convert the existing selection to Columnar. There's no need to cancel a selection; the selection mode can be toggled between Stream and Columnar at will.

5.244 Select Stream

Menu: Block > Select Stream

Default Shortcut Key: Alt+1

Macro function: SelectStream()

Stream Selection
The Select Stream command is used to set Boxer's *default selection mode* to Stream. In this mode text is selected by lines, as opposed to rectangular blocks as can be achieved with Select Columnar. Stream selection is the conventional type of text selection found in most text editors and word processors.

```
if (ListView1->Selected != NULL)
    {
    // build a file path to the selected macro file
    SourceFile = MacroDir + ListView1->Selected->Caption + MACRO_EXT;

    isedited = (MainForm->IsAnEditedFile(SourceFile) != NULL);

    // test to prevent unnecessary reloading of file
    if (isedited || (EditorFileName != ListView1->Selected->Caption))
        {
        EditorFileName = ListView1->Selected->Caption;
```

The *default selection mode* determines what selection style is used when text selection is initiated either from the keyboard or with the left mouse button. If the *default selection mode* is Columnar, a temporary override to Stream can be achieved by holding down the *Shift* key before selecting with the mouse.

Selecting Text with the Keyboard
Text selection can be performed by keyboard by pressing and holding down the *Shift* key and moving the text cursor. Any cursor movement key can be used to move the cursor, e.g., PgUp, PgDn, Home, End, etc, so long as the *Shift* key remains depressed. As the text cursor is moved through the file, the selected area is extended.

Selecting Text with the Mouse
Text selection can be accomplished with the mouse by pressing and holding the left mouse button and dragging the mouse to highlight the desired text. As the mouse pointer is moved through the file, the selected area is extended. An existing selection can be extended by depressing *Shift* or *Ctrl* before clicking or dragging the mouse.

Text can also be selected by clicking or dragging the mouse in the area to the left of column one. A single line can be selected by clicking the left mouse button at the far left edge of the line. Multiple lines can be selected by dragging the mouse in this area. An existing selection can be extended by depressing *Shift* before clicking in this area.

A word can be selected by double clicking anywhere within the word.

Selecting the Entire File
The entire file can be selected by issuing the Select All Text command.

Ending Text Selection
Text selection ends when the mouse button or the *Shift* key is released. The selected text remains highlighted and is ready to be operated upon by any of Boxer's Block commands, such as Cut, Copy, Append, Cut-Append, etc.

Extending a Selection with Other Commands
When text is selected, the Go to Line and Find commands provide options to extend the selection to the new cursor position that results from their operation. The Replace command provides an option to limit the replacement operation to the selected text.

Saving and Printing Text Selections

When text is selected, issuing either the Save or Save Selection As command prompts the user to save the selected area to a disk file. Similarly, when text is selected and the Print command is issued, an option is available to print only the selected area.

Deselecting Text
Selected text can be deselected by clicking once with the left mouse button, pressing the Escape key, or pressing any of the cursor arrow keys. Pressing the *Enter* key, or any other alphanumeric key will cause selected text to be deleted.


If a stream selection is started in the virtual space beyond the end of a line, or within the virtual space of a preceding Tab character, the starting column of the selection will be moved to the nearest column to the left which contains a character.


If you start a selection and find that the mode was set to Columnar, simply issue this command to convert the existing selection to Stream. There's no need to cancel a selection; the selection mode can be toggled between Stream and Columnar at will.

5.245 Select without Shift

Menu: Block > Select without Shift

Default Shortcut Key: Alt+M

Macro function: SelectWithoutShift()

This command can be used to initiate a text selection mode in which the *Shift* key does not need to be kept depressed during selection, as is the custom under Windows. After this command is issued, any cursor movement command can be used to extend the selection as desired. The text selection can be released by pressing *Escape*.

The type of selection that results is determined by the current selection mode, which can be set using the Select Columnar and Select Stream commands.


Former users of Boxer/DOS may welcome this command as it provides a method of text selection that approximates the operation of our earlier products.

5.246 Set Clipboard

Menu: Edit > Set Clipboard > Clipboard *n*

Default Shortcut Key: none

Macro function: SetClipboard()

The Set Clipboard command is used to set the active clipboard. The active clipboard can be either the Windows clipboard or one of Boxer's eight internal (private) clipboards. Text which is placed on the Windows clipboard will be accessible by other applications. Likewise, text placed on the Windows clipboard by other applications is

available to Boxer whenever the active clipboard is the Windows clipboard. Text that is placed on any of Boxer's internal clipboards is *not* available to other applications.

The content of each clipboard is displayed in a popup window as the menu cursor is moved across the clipboard's menu entry. This makes it easy to check what's on a clipboard without actually pasting the content into a file.

The content of Boxer's internal clipboards will be saved at the end of an edit session, as long as the length of the text on the clipboard is 2,048 characters or less. Because the content of the internal clipboards persists from session to session, and cannot be changed by other applications, these clipboards can be useful for storing frequently used text blocks for insertion into your files. The Edit Clipboard command might be used to create these text blocks and maintain them.

The content of a single clipboard can be cleared with the Clear Clipboard command. The content of *all* clipboards can be cleared with the Clear All Clipboards command.

When the content of a clipboard is displayed in a popup window, the text is displayed with an 8 point, fixed width, *Courier New* font. This font utilizes the ANSI character set mapping. If the current screen font uses an OEM character set mapping, and if characters outside the normal alphanumeric range reside on the clipboard, then the content of the clipboard may appear different in the popup window than it would in the underlying file. This difference is simply the result of a difference in character sets, and does not mean that the data on the clipboard has been adjusted or corrupted.

5.247 Set Clipboard Previous

Menu: Edit > Set Clipboard > Previous

Default Shortcut Key: none

Macro function: SetClipboardPrevious()

This command can be used to set the active clipboard to be the previous clipboard. For example, if clipboard 3 is active, issuing this command will make clipboard 2 the active clipboard. Clipboard 8 is considered the previous clipboard to the Windows clipboard.

5.248 Set Clipboard Next

Menu: Edit > Set Clipboard > Next

Default Shortcut Key: none

Macro function: SetClipboardNext()

This command can be used to set the active clipboard to be the next clipboard. For example, if clipboard 3 is active, issuing this command will make clipboard 4 the active

clipboard. The Windows clipboard is considered the next clipboard to the clipboard 8.

5.249 Shaded Tab Zones

Menu: View > Shaded Tab Zones

Default Shortcut Key: none

Macro function: ShadedTabZones()

The Shaded Tab Zones command toggles on and off a mode in which a different background screen color is used for alternating tab zones:

SAS	105.90	105.90	1.00	Bracket, power steeri
FFC,SAS	1.95	1.95	1.00	Crush nut, power stee
SAS	59.95	59.95	1.00	Idler arm, standard s
SAS	0.00	0.00	1.00	Idler arm, power stee
SAS	20.95	20.95	1.00	Bushing kit, idler ar
SAS	8.30	8.30	1.00	Bushing kit, idler ar
SAS	2.95	2.95	1.00	Idler arm seal kit, m
SAS	0.00	0.00	1.00	Steering gear box, re
SAS	0.00	0.00	1.00	Steering gear box, re
SAS	39.95	39.95	1.00	Steering column, hub,

This display mode is most helpful when Boxer is used to edit files containing character-separated field data, such as comma-separated values (CSV) or fixed-width field records. The Tab Display Size command would typically be used first to configure the proper tab stop settings for the data file. The *Intelli-Tabs* feature on that dialog is especially useful in this regard. Once the tab stop settings are entered, use this command to enable the shading of tab zones.

The alternative background color used by the Shaded Tab Zones command can be set using the Configure | Colors command. The tab zone is not depicted in the miniature screen on that dialog, but rather appears in the *Screen elements by name* listbox. Its name is *Tab Zone Background*.

5.250 Skip

Menu: Window > Skip -*or*- View > File Tabs > Skip File

Default Shortcut Key: none

Macro function: WindowSkip()

The Skip command can be used to mark a file/window so that it will be skipped over by Window Previous and Window Next when these commands are used to cycle through open files. The skip status of each file is stored when an edit session is closed, so it will

persist if the edit session is later resumed.

 The File Tab context menu also includes options to toggle the skip state for the current file, or to set or clear the skip status for all open files.

 Clicking on a file tab will cause that file's skip status to be released automatically, if the relevant option on the Configure | Preferences | Cursor dialog page is enabled.

5.251 Skip All

Menu: View > File Tabs > Skip All

Default Shortcut Key: none

Macro function: none

The Skip command can be used to mark a file/window so that it will be skipped over by Window Previous and Window Next when these commands are used to cycle through open files. The Skip All command sets the skip status of *all* open files to on. You might issue the Skip All command before loading new files for editing, thereby ensuring that the Window Previous and Window Next commands would cycle only within the newly opened files.

The skip status of each file is stored when an edit session is closed, so it will persist if the edit session is later resumed.

 The File Tab context menu also includes options to toggle the skip state for the current file, or to set or clear the skip status for all open files.

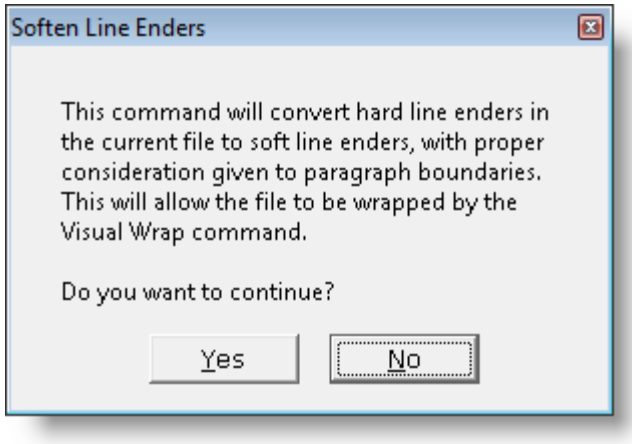
5.252 Soften Line Enders

Menu: Paragraph > Soften Line Enders

Default Shortcut Key: none

Macro function: SoftenLineEnders()

The Soften Line Enders command converts hard line enders to soft line enders, with proper consideration to paragraph boundaries. If a selection is present, the operation is restricted to the selected range of lines. If a selection is not present, the operation is performed across the whole file. A confirmation dialog will appear before the operation is performed:



The concept of "soft" and "hard" line enders relates to the Visual Wrap command. A line with one or more spaces at the end is considered to have a soft line ender. Lines without trailing spaces are considered to have hard line enders. When Visual Wrap mode is active, lines with soft line enders are eligible to be merged with the content of lines below, allowing text to be reformatted to fit within the window width (or whatever other wrapping margin is chosen).

Applying the Soften Line Enders command to a file has the effect of making the file flowable by Visual Wrap.

See also: Visual Wrap, Visual Wrap Options, Harden Line Enders

5.253 Sort File Tabs by Extension

Menu: View > File Tabs > Sort by Extension

Default Shortcut Key: none

Macro function: SortFileTabsByExt()

When checked, this menu option causes the File Tabs to be arranged alphabetically first by file extension, and then by filename.

Note: repositioning file tabs by drag-and-drop necessitates that any file tab sorting mode (name, extension or use) which may be in force be abandoned. Otherwise, when a new file is opened and the file tabs are resorted, the drag-and-drop ordering would be lost.

5.254 Sort File Tabs by Name

Menu: View > File Tabs > Sort by Name

Default Shortcut Key: none

Macro function: SortFileTabsByName()

When checked, this menu option causes the File Tabs to be arranged alphabetically by filename.

Note: repositioning file tabs by drag-and-drop necessitates that any file tab sorting mode (name, extension or use) which may be in force be abandoned. Otherwise, when a new file is opened and the file tabs are resorted, the drag-and-drop ordering would be lost.

5.255 Sort File Tabs by Use

Menu: View > File Tabs > Sort by Use

Default Shortcut Key: none

Macro function: SortFileTabsByUse()

When checked, this menu option causes the File Tabs to be arranged according to frequency of use. When a File Tab is clicked, the file is promoted to the first position.

Switching windows by keyboard will not cause the active file tab to be promoted to the first position. This only occurs when the file tab is clicked with the mouse.

Note: repositioning file tabs by drag-and-drop necessitates that any file tab sorting mode (name, extension or use) which may be in force be abandoned. Otherwise, when a new file is opened and the file tabs are resorted, the drag-and-drop ordering would be lost.

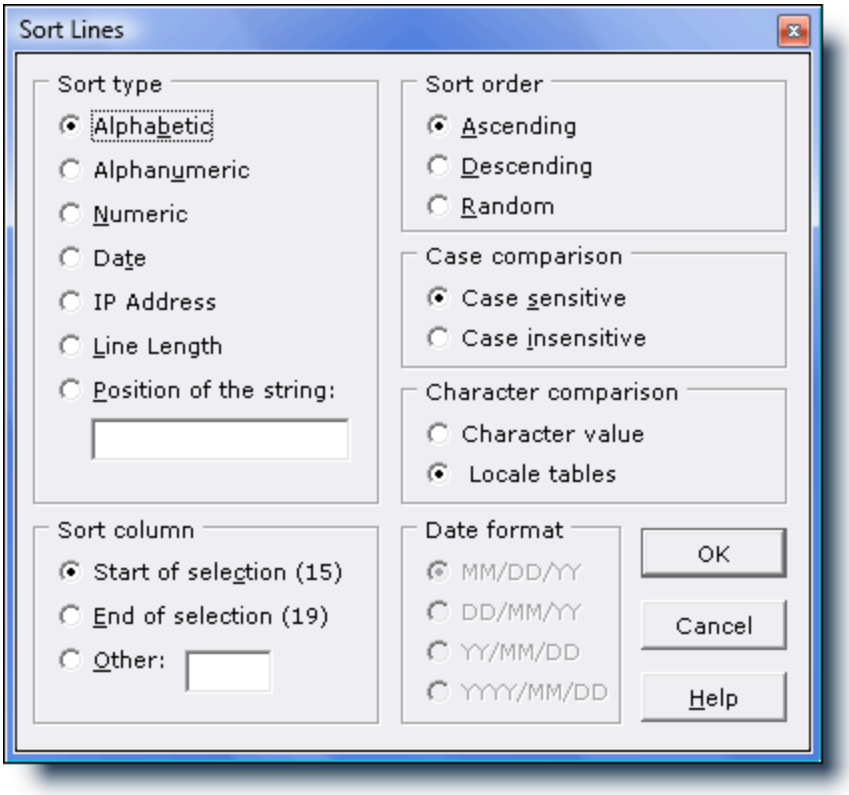
5.256 Sort Lines

Menu: Block > Sort Lines

Default Shortcut Key: none

Macro function: SortLines()

The Sort Lines command can be used to sort a range of selected lines. A variety of options are provided to control the nature of the sort performed. These options are described below:



Sort Type

Alphabetic
Use this option to sort alphabetically. Any digits that appear within the data will not be treated as numeric values.

Alphanumeric
This option sorts alphabetically, but embedded sequences of digits within the data are treated as numeric values. This is sometimes called a *natural* sort, and is best understood with an example. Consider the results of the Alphabetic and Alphanumeric sorting options when applied to a list of filenames:

Alphabetic sort:	**Alphanumeric sort:**
z1.txt	z1.txt
z10.txt	z2.txt
z100.txt	z3.txt
z101.txt	z4.txt
z102.txt	z5.txt
z11.txt	z6.txt
z12.txt	z7.txt
z19.txt	z8.txt
z2.txt	z9.txt
z20.txt	z10.txt
z3.txt	z11.txt
z4.txt	z12.txt
z5.txt	z19.txt

```
z6.txt                  z200.txt
z7.txt                  z100.txt
z8.txt                  z101.txt
z9.txt                  z102.txt
```

In the left column, the results of an Alphabetic sort are shown. On the right, the more pleasing results of an Alphanumeric are displayed.

Numeric
Use this option to sort numerically. The data will be interpreted as numbers, and not as character data. Alphabetic data will sort as though its value were zero.

Date
Use this option to sort chronologically by date. Be sure to set the proper Date Format in the options box provided.

IP Address
Use this option to sort IP Addresses data with proper consideration to each node within the address.

Line Length
Use this option to sort lines according to their length.

Position of the string:
This option allows data to be sorted based on the position of a supplied string within the data. This option can be useful for segregating lines of data that contain a certain type of information. For example, by sorting on the string '@', lines containing email addresses would be isolated from those lines not containing email addresses.

Sort Column

Start of selection
Use this option if the sort should be performed based on the starting column of the selection. The column number of the start of the selection is shown in parentheses to the right.

End of selection
Use this option if the sort should be performed based on the ending column of the selection. The column number of the end of the selection is shown in parentheses to the right.

Other
Use this option if the sort should be performed based on some other column in the data.

Sort Order

Ascending
Use this option to sort in increasing order.

 The sort command will consult the current locale so that accented characters are

sorted according to the local collating sequence.

Descending
Use this option to sort in decreasing order.


The sort command will consult the current locale so that accented characters are sorted according to the local collating sequence.

Random
Use this option to sort randomly. This option might be used to randomly order a list which was already sorted. When this option is selected, all other options on the dialog become irrelevant.

Case Comparison

Case Sensitive
When an alphabetic or alphanumeric sort is being performed, this option can be used to ensure that character case is considered significant.

Case Insensitive
When an alphabetic or alphanumeric sort is being performed, this option can be used to ensure that character case is ignored.

Character Comparison

Character Value
When an alphabetic or alphanumeric sort is being performed, this option causes characters to be compared based on their actual character values. This is sometimes called an 'ASCII sort.'

Locale Tables
When an alphabetic or alphanumeric sort is being performed, this option causes characters to be compared using the 'locale tables' supplied by the operating system. Using the locale tables ensures that when accented characters are encountered in the data being sorted, they will be sorted according to local custom.

The results of a Locale Table sort can vary from those achieved using Character Value. A case sensitive sort using Character Value would yield the following result:

```
AAA
BBB
CCC
aaa
bbb
ccc
```

If the same data is sorted with case sensitive and Locale Tables, the following result is achieved:

```
aaa
AAA
bbb
```

BBB
ccc
CCC

☞ A Locale Table sort uses a "word sort," rather than a "string sort." A word sort treats hyphens and apostrophes differently than it treats other symbols that are not alphanumeric, in order to ensure that words such as "coop" and "co-op" stay together within a sorted list.

Date Format

The date options below are applicable when a Date sort is being performed. The slash character is shown for illustration only; any separator symbol--or none at all--may appear in the data being sorted.

MM / DD / YY
Data is formatted with 2-digit month, date and year. This option can also be used for MM / DD / YYYY format dates.

DD / MM / YY
Data is formatted with 2-digit date, month, and year. This option can also be used for DD / MM / YYYY format dates.

YY / MM / DD
Data is formatted with 2-digit year, month, and date.

YYYY / MM / DD
Data is formatted with 4-digit year, 2-digit month and 2-digit date.

5.257 Spaces to Tabs

Menu: Block > Convert Other > Spaces to Tabs

Default Shortcut Key: none

Macro function: SpacesToTabs()

The Spaces to Tabs command can be used to convert the Spaces within a selected area of text into an equivalent number of Tabs. In doing so, Boxer uses the current display value of a Tab (View | Tab Display Size) to determine how many Tabs should be used.

☞ No attempt is made to determine whether the Spaces being changed reside within a quoted string. Programmers should be careful when using this command, as changing Spaces within a quoted string may yield undesirable results if the string was to be displayed in a message on-screen.

☞ The Spaces to Tabs and Tabs to Spaces commands are not opposites. If the Tab Display Size is 4, the sentence:

```
The<tab>dog<tab>ran<tab>wild.
```

would be converted by the Tabs to Spaces command to:

```
The<space>dog<space>ran<space>wild.
```

Running the Spaces to Tabs command on this sentence would not yield the original sentence. A sequence of two or more spaces is required before a tab character is considered for placement into the converted text.

5.258 Spell Checker

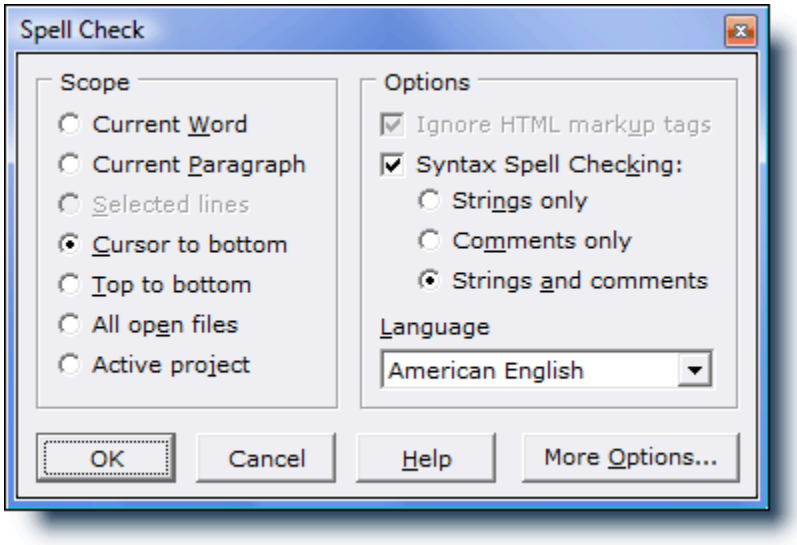
Menu: Tools > Spell Checker

Default Shortcut Key: F7

Macro function: SpellChecker()

The Spell Checker command provides access to the built-in spell checking facility. Boxer's spell checker provides several options to control the scope of the spell check operation, and some unique options which permit spell checking to be performed within program source code files.

For information about Boxer's on-the-fly spell checking feature, which checks text as you type, see the Active Spell Checking command.



Scope

Current Word
Use this option if only the word at the text cursor is to be checked.

Current Paragraph

Use this option to spell check the current paragraph only.

Selected lines
Use this option to constrain the spell check to the selected text. When text is not selected, this option will be disabled.

Cursor to bottom
Use this option to spell check from the cursor to the end of file.

Top to bottom
Use this option to spell check the entire file.

All open files
Use this option to spell check all open files.

Active project
Use this option to limit the scope of the spell check to those files within the active project.

Options

Ignore HTML markup tags
When an HTML file is being edited, this option can be used to tell the Spell Checker to ignore HTML markup tags, thereby avoiding the false errors which would occur if the entire file were checked.

Syntax Spell Checking
When editing a file for which Boxer has Syntax Highlighting information, this option can be used to tell the Spell Checker to check only that text which matches a designated syntax. This makes it possible to spell check program code without getting false hits for reserved words, variable names, and other text which naturally cannot appear within a dictionary. When this box is checked, the options below become available for selection.

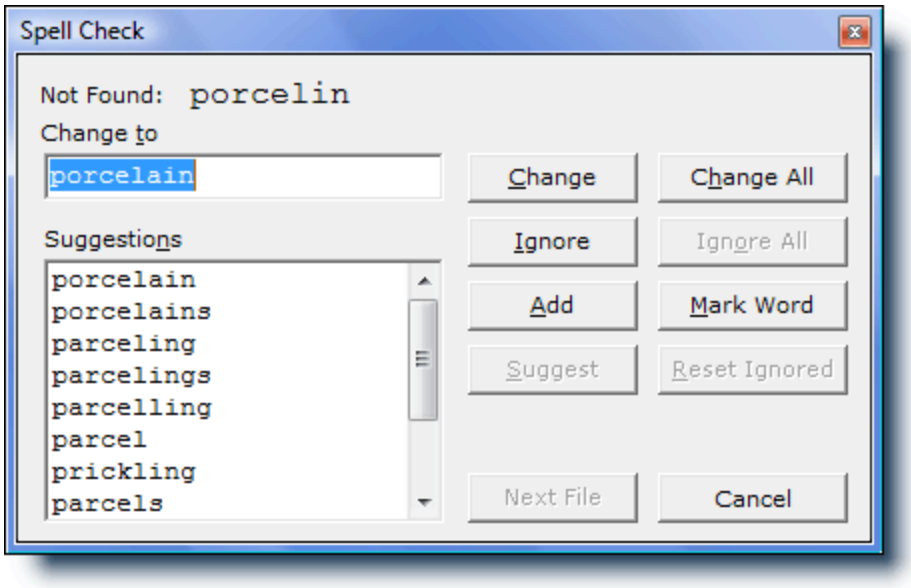
> **Strings only**
> Use this option to spell check only that text which appears in String context. Boxer uses its Syntax Highlighting information to determine the syntax of the text being checked.
>
> **Comments only**
> Use this option to spell check only that text which appears in Comment context. Both block comment text and end-of-line comment text will be checked. Boxer uses its Syntax Highlighting information to determine the syntax of the text being checked.
>
> **Strings and Comments**
> Use this option to spell check only that text which appears in either String or Comment context. Both block comment text and end-of-line comment text will be checked. Boxer uses its Syntax Highlighting information to determine the syntax of the text being checked.

Language

Use this option to select the dictionary to be used by the Spell Checker. Dictionaries are available for the following languages: American English, British English, Dutch, French, German, Italian, Czech and Spanish. Three specialty dictionaries are also available: Legal, Medical and 'Moby', an extra large American English dictionary. If Boxer was not supplied with the dictionary you prefer to use, please visit our website at www.boxersoftware.com to obtain our other dictionaries. The dictionaries are located in the Downloads area.

More Options...
The *More Options* button provides a quick way to summon the Configure | Preferences | Other dialog page, which contains additional Spell Checker options.

After specifying the scope and other options, the Spell Check operation begins. When a word is encountered which is not found in the dictionary, a popup dialog box is presented:



The position of the popup box is selected so as not to overlap the offending word within the editor window. The word that was not found is displayed in bold, and is also highlighted in the editor window to show its context. Several options are available:

Change to
The *Change to* edit box displays a suggested word which the offending word might be replaced with. The text in this box can be edited as may be needed.

☞ To ensure that special characters are displayed in the *Change to* box as they will appear when inserted into the editor, the *Change to* box uses the same font as is used in the editor itself.

Suggestions

The *Suggestions* listbox displays a list of other words which might be used to replace the offending word. Click on a word to select it and move it into the *Change to* edit box.

Change
Use the *Change* button to replace the offending word with the word in the *Change to* box.

Change All
Use the *Change All* button to replace the offending word with the word in the *Change to* box, and to indicate that all future occurrences of the word should also be changed.

Ignore
Use the *Ignore* button to skip the offending word. Future occurrences of the word will be presented when they occur.

Ignore All
Use the *Ignore All* button to skip the offending word, and to indicate that all future occurrences should also be ignored.

Add
Use the *Add* button to add the offending word to the dictionary. Words which are added to the dictionary are saved within the file `userdict.txt` in Boxer's *data folder*. This file can also be edited within Boxer to add other words, or to remove words which may have been added mistakenly.


Words which are added to the user dictionary will be accepted as correctly spelled words in any case configuration in which they may occur. For example, if the word `ebay` is added to the dictionary, it will be accepted in any of the following forms: `eBay`, `ebAy`, and `ebaY`. This liberal processing was necessary because the third-party dictionary that Boxer uses is not processed in a case sensitive manner. Before this handling was put in place, the word `eBay` would always be reported as misspelled, even when `eBay` (or any variant) had been added to the user dictionary.

Mark Word
Use the *Mark Word* button to surround the offending word with pound signs (###). This makes it easy to locate the word later on to make manual adjustments. When the Spell Check operation is complete, the cursor will be placed on the first marked word.

Suggest
Once a change has been made to the word in the *Change to* box, the *Suggest* button can be used to fill the *Suggestions* listbox with other words which may be related to the word.

Reset Ignored
Use the *Reset Ignored* button to clear the list of ignored words which has been built during the current edit session.

Next File
Use the *Next File* button to skip the rest of the current file and move to the next file to be checked.

By default, Boxer is configured to use the American English dictionary. Dictionaries are also available for British English, Dutch, French, German, Italian, Czech and Spanish. Three specialty dictionaries are also available: Legal, Medical and 'Moby', an extra large American English dictionary. The active dictionary can be set on the dialog that appears when the Spell Checker is first run. If Boxer was not supplied with the dictionary you prefer to use, please visit our website at www.boxersoftware.com to obtain our other dictionaries. The dictionaries are located in the Downloads area.

Several options which can be used to further control the Spell Checking process are available on the Configure | Preferences | Other options page.

5.259 Split Horizontal

Menu: Window > Split Horizontal

Default Shortcut Key: none

Macro function: SplitHorizontal()

The Split Horizontal command can be used to split the current window horizontally. A split window provides a second view into the same file, allowing two different sections of the file to be viewed simultaneously in different window *panes*. Each window pane can be scrolled separately from the other, just as if a second window were in use.

After a window is split, a thin *splitter bar* appears which visually separates the two panes. The splitter bar can be dragged up or down with the left mouse button to resize the window panes.

Clicking on the splitter bar with the right mouse button provides access to its context menu. The context menu has options to change the split from horizontal to vertical, or to turn off the horizontal split so the window becomes whole again.

The Window Next command can be used to move from the top pane to the bottom pane, while the Window Previous command changes *focus* from the bottom pane to the top.

When the height of the window is increased or decreased due to window resizing, the relative position of the window split will be maintained, so long as each pane remains taller than the minimum window height.

If the Column Ruler is in use, it will appear only in the top window pane of a horizontally split window.

When the panes of a split window are resized with the mouse, a report appears on the status line that shows the percentage of width/height allocated to each pane.

5.260 Split Vertical

Menu: Window > Split Vertical

Default Shortcut Key: none

Macro function: SplitVertical()

The Split Vertical command can be used to split the current window vertically. A split window provides a second view into the same file, allowing two different sections of the file to be viewed simultaneously in different window *panes*. Each window pane can be scrolled separately from the other, just as if a second window were in use.

After a window is split, a thin *splitter bar* appears which visually separates the two panes. The splitter bar can be dragged left or right with the left mouse button to resize the window panes.

Clicking on the splitter bar with the right mouse button provides access to its context menu. The context menu has options to change the split from vertical to horizontal, or to turn off the vertical split so the window becomes whole again.

The Window Next command can be used to move from the left pane to the right pane, while the Window Previous command changes *focus* from the right pane to the left.

When the width of the window is increased or decreased due to window resizing, the relative position of the window split will be maintained, so long as each pane remains wider than the minimum window width.

If the Column Ruler is in use, it will appear in both the left and right window panes of a vertically split window.


When the panes of a split window are resized with the mouse, a report appears on the status line that shows the percentage of width/height allocated to each pane.

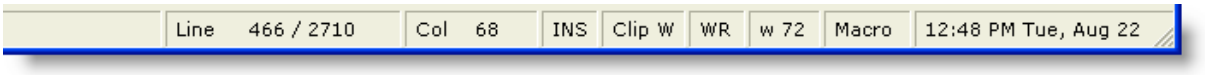
5.261 Status Bar

Menu: View > Status Bar

Default Shortcut Key: none

Macro function: ViewStatusBar()

The View Status Bar command is used to toggle on and off the display of Boxer's *Status Bar* which appears at screen bottom. The status bar displays the location of the text cursor in the current file, the current edit mode, current Clipboard, read-only state, Typing Wrap and Text Width settings and the current time and date.



The leftmost area of the Status Bar is used to present various information depending on the command being performed. For example, while text is being selected, a report is displayed that shows the number of lines and characters selected. The percentage of the selection, with respect to the whole file, is also displayed.

Double clicking in each of the status fields is recognized as a shorthand method of issuing a related command.

The Line Number field displays the current line number in the current file. Double clicking in the Line Number field will issue the Go to Line command. In Visual Wrap mode, the Line Number field will display paragraph numbers, since a one-to-one relationship between physical lines and screen lines no longer exists. In Visual Wrap mode, double clicking in this field will issue the Go to Paragraph command.

The Column Number field displays the current column number in the current file. Double clicking in the Column Number field will issue the Go to Column command.

The Edit Mode field indicates the current edit mode. 'INS' denotes Insert mode. 'TYP' denotes Typeover mode. Double clicking in the Edit Mode field will toggle the edit mode between Insert and Typeover modes.

The Clipboard field displays the active clipboard. 'W' indicates the Windows clipboard; internal clipboards are denoted by the digits 1-8. Double-clicking in this field advances the active clipboard by one. Shift double-clicking decreases the active clipboard by one.

To the right of the Clipboard field is the Read-Only field. If the current file is being viewed in read-only mode, 'RO' is displayed. If the file is eligible for changes, 'WR' is displayed. Double-clicking in this field will change the state of the current file within the editor. If a file is being edited in read-only mode because its on-disk read-only file attribute is set, an option is provided to change the file's on-disk read-only attribute. Changing a writable file to read-only mode does not alter a file's on-disk file attribute.

To the right of the Read-Only field is the Typing Wrap and Text Width field. Double clicking atop the 'w' in this field will toggle Typing Wrap mode on and off. A lowercase 'w' denotes off; an uppercase 'W' denotes on. Double clicking in the numeric portion of this field will issue the Text Width command.

The macro field serves several purposes. When the word 'Macro' is not flashing, double-clicking in this field will display the Macro dialog. When a macro is running, the word 'Macro' will flash intermittently. When keystrokes are being recorded using the Record Keys command, the macro field will flash the word 'Record'.

At the far right of the Status Bar is the Time and Date display. Double clicking atop the time display will issue the Insert Short Time command. Double clicking atop the date display will issue the Insert Short Date command.

Due to a problem reported by users in countries that do not use the Western/Latin *code page*, the date in the lower right corner of the status bar will now be displayed

in English, and not in the language dictated by the operating system's regional settings. The Insert Short/Long Time/Date commands will continue to honor the system's regional settings.

Unless screen space is at a premium, it is recommended that the Status Bar display be left on. Right clicking on the Status Bar summons its context menu, which allows it to be turned off.

5.262 Strip HTML/XML Tags

Menu: Block > Strip HTML/XML Tags

Default Shortcut Key: none

Macro function: StripHTMLTags()

The Strip HTML/XML Tags command can be used to remove HTML or XML tags from selected text. HTML tags are markup sequences which appear within the '<' and '>' characters. Boxer does not require that the tag names found within these brackets be legitimate HTML tags. It merely removes any text found to be within such delimiters. In this way, Boxer will be able to process new tags properly as the HTML standard evolves.

Caution: If the text being processed contains unbalanced angle bracket characters--specifically an unmated open angle bracket--then all text following the open angle bracket will be treated as an HTML tag, and will be removed.

In addition to stripping HTML tags, the following HTML sequences will be converted to their character equivalents:

	<space>
&	&
"	"
<	<
>	>
–	-
—	--
‘	'
’	'
“	"
”	"
…	...

The conversion of other such sequences is complicated by the fact that accented characters do not map to unique character codes in the ANSI and OEM characters set. These translations are therefore not performed.

5.263 Strip Leading Spaces

Menu: Block > Strip Leading Spaces

Default Shortcut Key: none

Macro function: StripLeadingSpaces()

The Strip Leading Spaces command can be used to remove leading Spaces and/or Tabs from the start of each line within the selection.

5.264 Strip Trailing Spaces

Menu: Block > Strip Trailing Spaces

Default Shortcut Key: None

Macro function: StripTrailingSpaces()

The Strip Trailing Spaces command can be used to remove trailing Spaces and/or Tabs from the end of each line within the selection. The number of characters removed is reported upon completion.

Trailing blanks can also be stripped automatically when **loading** a file. See the *Strip trailing blanks when loading a file* option on the Configure | Preference | File I/Ooptions page.

Trailing blanks can also be stripped automatically when **saving** a file. See the *Strip trailing blanks when saving a file* option on the Configure | Preference | File I/O options page.

5.265 Swap Lines

Menu: none

Default Shortcut Key: F4

Macro function: SwapLines()

The Swap Lines command exchanges the current line with the line below. The text cursor remains on the same line it was on before the command was issued. This command is useful for swapping the order of a pair of items within an ordered list.

The last line in the file is not eligible for this operation.


This command used to reside in the Edit menu, but has been replaced by the Move Line Up and Move Line Down commands. Though not accessible directly from the menu, the command remains active internally, and can be accessed by a key

assignment, and via its macro function.

5.266 Swap Words

Menu: Edit > Swap Words

Default Shortcut Key: Shift+F4

Macro function: SwapWords()

The Swap Words command can be used to swap the word at the text cursor with the word to its right.

The characters which serve to delimit words can be set on the Configure | Preferences | Cursor options page. The option is titled *These characters will delimit words*.

The last word on a line cannot be swapped, since the Swap Words command does not span lines.

5.267 Synchronized Scroll

Menu: View > Synchronized Scroll

Default Shortcut Key: none

Macro function: none (the interactive nature of this command makes it unsuitable for use within a macro)

The Synchronized Scroll command can be used to enter a display mode in which all open windows will scroll synchronously. This command is useful for hands-off file browsing, or for comparing similar files in side-by-side windows.

The initial direction of scrolling is downward, but the *Up* and *Down* arrow keys can be used to change direction at any time.

The *Left Arrow* and *Right Arrow* keys can be used to decrease or increase the scrolling delay, respectively.

Pressing *Right Arrow* repeatedly through the range of delay settings will set the delay to infinite. The infinite setting effectively locks all open windows to one another. The *Up Arrow* and *Down Arrow* keys can then be used to scroll all windows synchronously.

The *Home* and *End* keys can be used to move quickly to the minimum and maximum (infinite) delay settings.

Scrolling can be canceled with the *Esc* key or by pressing any key other than the arrow keys.

☞ You may observe that Synchronized Scrolling quickens when the mouse is being moved. This is because a program receives more CPU cycles from the operating system when it is perceived to be active than when the operating system believes the program to be idle.

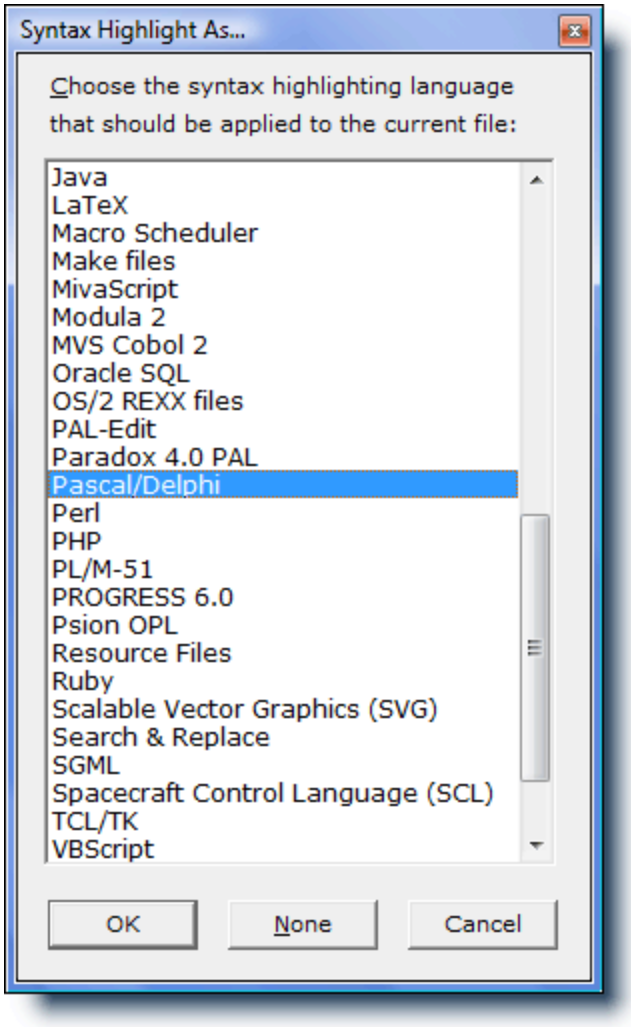
5.268 Syntax Highlight As

Menu: View > Syntax Highlight As

Default Shortcut Key: none

Macro function: SyntaxHighlightAs()

The Syntax Highlight As command provides a means to override the syntax highlighting that occurs due to a file's extension, or to select a language for a file that would not otherwise be eligible for highlighting. For example, if you're viewing a file named `index.html.bak`, the Syntax Highlight As command would allow HTML to be designated as the syntax highlighting language, even though the file's `.bak` extension is not configured for HTML highlighting.



The *None* button allows a file to be disassociated from its syntax highlighting language, without the need to disable syntax highlighting for all files being edited, as the View | Syntax Highlighting command can do.

The duration of the Syntax Highlight As assignment is for the current editing session only. To permanently associate a file type with a syntax highlighting language, use the Configure | Syntax Highlighting command to add its file extension to the list of recognized extensions.

5.269 Syntax Highlighting (Configure)

Menu: Configure > Syntax Highlighting

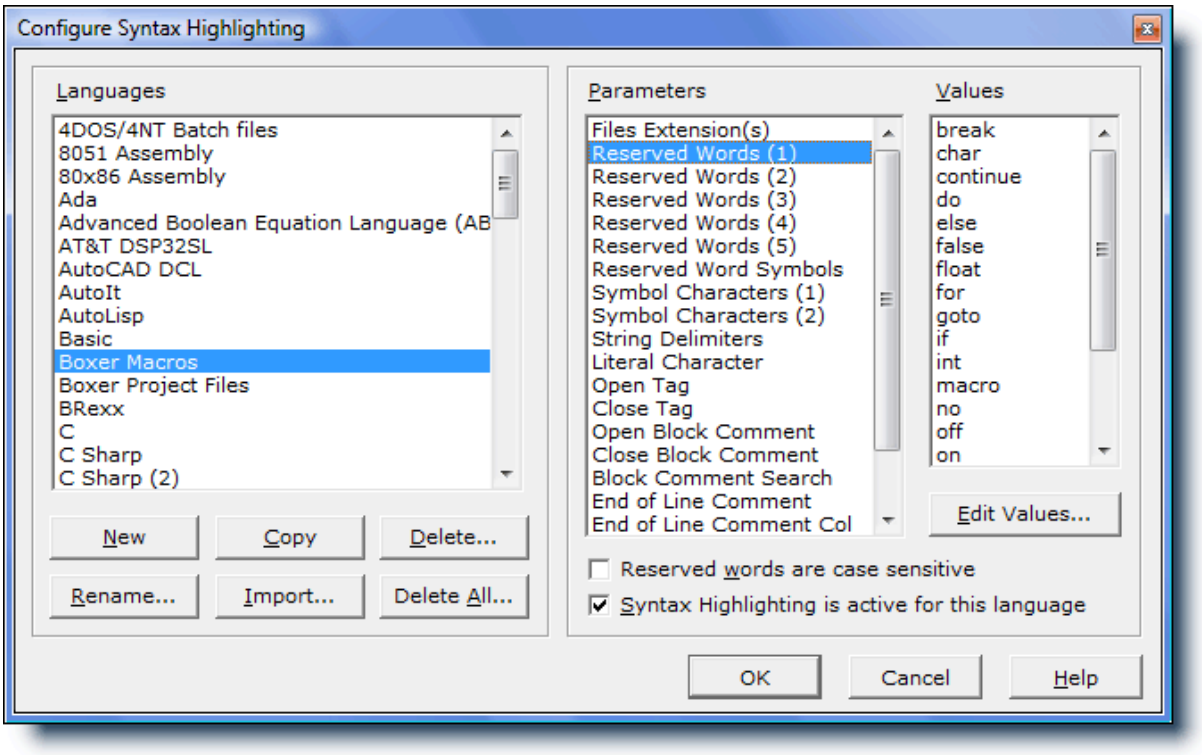
Default Shortcut Key: none

Macro function: ConfigureSyntaxHighlighting()

The Configure Syntax Highlighting command is used to specify the information needed

by Boxer to perform on-screen Color Syntax Highlighting and Color Syntax Printing and for its other syntax-related features. Boxer is supplied with pre-defined syntax information for many popular languages. The Configure Syntax Highlighting command can be used to edit any of the pre-defined syntax information, or to define the syntax for new languages.

The dialog box below is used to define syntax information:



Boxer determines whether to perform Syntax Highlighting on the current file according to three factors.

- The extension of the current file must be one for which syntax information has been defined
- The language must be configured to be 'active' on the Configure Syntax Highlighting form (see the *Syntax Highlighting is active for this language* checkbox)
- Syntax Highlighting must be enabled on the View menu

Languages

New Button

Use the *New* button to create a Syntax Highlighting entry for a new language. An entry called *New Language* will be created which can be changed with the *Rename* button. Up to 100 different languages can be defined.

Rename Button

Use the *Rename* button to change the name of the selected language. A pop-up dialog box will appear for entering the new name. The name used will not appear anywhere other than in the list of languages.

Copy Button
Use the *Copy* button to create a copy of the currently selected language entry. The name '*Copy of...*' will be used and can be changed with the *Rename* button. The *Copy* function can save time when defining a new language which has similar characteristics to an existing language.

Import Button
The *Import* function can be used to import syntax information in the format used by our BOXER/DOS, BOXER/TKO and BOXER/OS2 products. If you have created custom syntax information with our other editors, that information can be imported directly by Boxer for Windows. First isolate the syntax information blocks which are to be imported by copying them from the `DEFAULT.CFG` file into a temporary file. Then click the *Import* button and specify that file as the name of the file to be imported. Boxer will read the named file and automatically convert the syntax information into the new format. Because the old information format did not contain a name field, you will be prompted during conversion to supply a name for each language as it is imported.

The *Import* function can also be used to import a syntax information block which has been extracted from the `Syntax.ini` file, in which Boxer stores its syntax information. This procedure may be useful for passing syntax information from PC to PC or for installing new syntax information files as they become available from Boxer Software.

Delete Button
Use the *Delete* button to delete the currently selected language. A confirmation is required before the deletion occurs. If a language is accidentally deleted, you can recover it by using the *Cancel* button.

If you simply wish to disable Syntax Highlighting for a particular language, use the *Syntax Highlighting is active for this language* checkbox described below.

Delete All Button
Use the *Delete All* button to delete **ALL** languages. A confirmation is required before the deletions will occur. You can recover from an accidental deletion by using the *Cancel* button.

USE THIS COMMAND ONLY IF YOU WISH TO DELETE ALL SYNTAX INFORMATION.

If you wish to disable syntax highlighting for all languages, use the *Perform Syntax Highlighting* option on the Configure | Preferences | Display options page. That option is non-destructive.

Parameters

The *Parameters* listbox contains all of the parameters which can be defined for a given language. Each of these parameters is discussed below:

File Extension(s)
This parameter is used to designate the file extensions which belong to the language being defined. The file extensions are named one per line, with a leading period (`.`). Be sure to include all file extensions for which highlighting is desired, such as header files, and include files. If a file type commonly goes by two names, such as `.HTM` and `.HTML`, be sure to include both extensions to guarantee that highlighting will be performed on all files desired.

☞ To designate that highlighting is to be applied to files without an extension, use a lone period (`.`) on a line.

Reserved Words 1, 2, 3, 4, 5
These parameters are used to list the reserved words (sometimes known as *keywords*) which are to be highlighted. Reserved words are entered one word per line. No care need be taken to preserve an alphabetic sort, since sorting is performed automatically by Boxer.

If reserved words are to be considered case-sensitive, they should be entered in the case which is recognized by the language.

Boxer permits up to 5 sets of reserved words to be defined, and each set can be distinctly colored (see Configure | Colors). *Reserved Words 1* might be used for language keywords, such as `for`, `if`, `while`, `loop`, etc. *Reserved Words 2* might be used for preprocessor directives such as `#include`, `#define`, `#ifdef`, etc. *Reserved Words 3* might be used for library functions such as `strcpy`, `strlen`, `strcat`, etc. The *Reserved Words 4* and *Reserved Words 5* groups provide flexibility for coloring other classes of words.

☞ The *Reserved Words 4* and *Reserved Words 5* groups do not have their own sample text entries in the miniature configuration screen in the Configure | Colors dialog. To assign colors and styles to these screen elements, select them from the *Screen elements by name* listbox.

☞ The wildcard characters '?' and '*' are no longer recognized when defining reserved words as they were in our earlier products. We found that very few languages need this feature, while some popular languages (such as Perl) need to use '?' and '*' within their reserved words.

Reserved Word Symbols
This parameter is used to designate those symbols which are permissible within a reserved word or user variable, so that Boxer does not mistakenly highlight a phrase which happens to begin with a reserved word. An example will help clarify:

If 'read' is a reserved word, and you want to ensure that the first four letters of a variable named 'read_my_data_file' are not mistakenly highlighted as a reserved word, designate the underscore in the *Reserved Word Symbols* parameter. This tells Boxer that the underscore is allowed to appear in a reserved word or user variable, and that it is *not* a valid separator.

Alphanumeric characters are automatically permissible within reserved words. Add any

additional characters which require similar treatment, one per line.

Symbol Characters 1, 2
These parameters are used to designate those characters which are to receive Symbol coloration. Two different sets of symbols are permitted, providing extra flexibility for color combinations. Designate one symbol per line.

String Delimiters
This parameter is used to designate the character(s) which are used to delimit strings. These characters vary from language to language, but are typically the double quote and/or single quote characters. Designate one symbol per line.


Boxer does not support the highlighting of strings that extend across more than one line. If you must highlight such strings, and if the language in question uses opening and closing string delimiters that are unique to one another, then you may wish to define these sequences as though they were Block Comments. Strings would then be colorized in Comment color, but multi-line strings would then be handled.

Literal Characters
This parameter is used to designate the character which is used to remove significance from an opening or closing *String Delimiter* character while within a string. Typically this is the backslash (\) character.

Open Tag
This parameter is used to designate the character which opens a tag for languages such as HTML, XML and SGML. These languages differ from conventional programming languages in that all 'code' within the file appears within markup tags, and all text outside of markup tags is considered to be the text of the document. For all other conventional programming languages, this parameter should be left blank.

Close Tag
This parameter is used to designate the character which closes a tag for languages such as HTML, XML and SGML. These languages differ from conventional programming languages in that all 'code' within the file appears within markup tags, and all text outside of markup tags is considered the text of the document. For all other conventional programming languages, this parameter should be left blank.

Open Block Comment
This parameter is used to designate the sequence (or sequences) which are used to open a multi-line block comment. Place each sequence on its own line.

Close Block Comment
This parameter is used to designate the sequence (or sequences) which are used to close a multi-line block comment. Place each sequence on its own line.

Block Comment Search
In order to properly handle multi-line comment blocks, Boxer must at times search backward in the current file to determine if a multi-line comment remains open from a line which is off-screen. This parameter designates the number of lines which should be searched during this effort. Higher values will result in better display accuracy when

large block comments are used, but can slow screen display at other times.

End of Line Comment
This parameter is used to the designate the sequence (or sequences) which are used to open an end-of-line comment. An end-of-line comment persists from the point it is opened until the end-of-line. Place each *End of Line Comment* sequence on its own line.


For each *End of Line Comment* defined, a corresponding *End of Line Comment Column* must also be defined**.** See the paragraph immediately below for details.


If an *End of Line Comment* sequence includes a Space character, you'll find that comments in your text will not be not be colorized when the View Visible Spaces option is in use. This occurs because the Space character in the *End of Line Comment* sequence does not match the value of the visible space character used on screen. You can remedy this by adding a duplicate sequence that uses the visible space character in place of the Space. You can find the value of the visible space character on the Configure | Preferences | Display dialog page. This character must be entered into the edit dialog with a special technique; see the Help topic Inserting Special Character for details. Finally, remember to add the accompanying *End of Line Comment Column* parameter to mate with the duplicate *End of Line Comment* sequence.

End of Line Comment Column
This parameter is used to designate the column in which an associated *End of Line Comment* should be recognized. Some languages require that an *End of Line Comment* sequence be recognized only when it appears in a particular column, such as column 1 or column 7.

Enter the required column value, or enter 0 (zero) if the *End of Line Comment* sequence is to be recognized in all column positions. When multiple *End of Line Comment* sequences have been defined, each sequence must have a corresponding *End of Line Comment Column* entry, in the same list position as its mate.

Languages such as Clipper, dBase and FoxPro require that the asterisk (*) be recognized as an end of line comment when the symbol appears as the first non-blank character in the line. In other contexts the asterisk must retain its conventional meaning as the multiply symbol. This logic can be requested in Boxer (for the asterisk or any other *End of Line Comment* sequence) by using a value of -1 for the *End of Line Comment Column* parameter.

Tab Stops
Use this parameter to designate tab stop settings for files matching the *File Extensions* parameter of this language configuration. See the View | Tab Display Size command for more information about variable width tab stops.

Help File
This parameter can be used to designate an associated Windows help file (`.HLP` or `.CHM`) for the language being defined. Once the help file has been defined for a language, context-sensitive help for the word beneath the text cursor can be obtained by issuing the Help command, which is ordinarily assigned to *F1*. To obtain Boxer's

native Help instead of language-specific help, simply move the text cursor into an open area of text before requesting Help. The full filepath to the reference document must be supplied.

This parameter can also be used to designate an HTML-format reference file, or indeed *any* type of reference document which the operating system knows how to open based on its file extension. For example, if you have a Microsoft Word `.DOC` file or Adobe Acrobat `.PDF` file that details the syntax of a language, these too can be named in the *Help File* parameter for that language.


The ability to display context-sensitive help for the word beneath the text cursor is available only when launching WinHelp `(.HLP)` and HTML Help (`.CHM`) files, and not when `.HTML, .PDF, .DOC` and other files are used.

Syntax Spell
This parameter is used to control how the Active Spell Checking feature should be applied to files which are syntax highlighted. A value of 0, 1, 2 or 3 can be used, with the effect being as follows:

0: Active Spell Checking will not be performed when editing syntax highlighted files
1: Active Spell Checking will be performed only within comments and quoted strings
2: Active Spell Checking will be performed within comments, quoted strings and 'normal' text
3: Active Spell Checking will be performed only on 'normal' text

Reserved Words are case sensitive
Use this option to designate whether the reserved word lists should be treated as case-sensitive. If this option is checked, a reserved word must match a list entry exactly in order to be highlighted. If this option is not checked, a reserved word will match a list entry even when its case is different.

This option should be selected to correspond to the requirements of the language being defined, so that Boxer can provide accurate visual feedback when a reserved word has been mistyped.

Syntax Highlighting is active for this language
Use this option to enable or disable highlighting for the current language. This option is the simplest way to disable syntax highlighting for a single language. One reason to disable a language would be to cure a file extension conflict with another language.


Use the View | Syntax Highlighting command to quickly disable syntax highlighting for *all* languages.

Notes and Tips


In addition to on-screen Syntax Highlighting, the language information defined with this command is also used for the following commands and features:

- Color Syntax Printing
- Monochrome Syntax Printing

- Syntax Spelling
- Active Spell Checking
- Auto-Complete
- Syntax Matching
- Comment
- Uncomment

If you define syntax information for new languages, or if you make additions or corrections to the pre-defined languages, please consider sending your information to us. This will allow us to keep our information current, and make it available to other Boxer users. Syntax information can be sent to support@boxersoftware.com. Thank you in advance for your contributions.

Some users have reported using Syntax Highlighting as a teaching aid for young readers. One customer told of how she had created a syntax definition in which common nouns, verbs and adjectives were assigned to three of Boxer's reserved word classes. Then, when a file with the required file extension was displayed, each part of speech would be highlighted in its own color. Another user reported creating a 'language' definition so that headings within a dense parts list would be highlighted in color. As you can see, the uses for Syntax Highlighting extend far beyond its utility to programmers.

The highlighting of Java and Active Server code poses special problems for Boxer. These languages can include HTML markup tags as well as sections of conventional procedural style code. At times the open angle bracket (<) is a less-than symbol, at other times it could open an HTML markup tag. A rigorous handling of Java code would require that a language parser be used, which is not the method by which Boxer's (general purpose) highlighting is performed. Therefore, Boxer's default syntax information for Java has been designed to highlight Java program code, but not to highlight any HTML markup tags which might appear therein.

5.270 Syntax Highlighting (View)

Menu: View > Syntax Highlighting

Default Shortcut Key: none

Macro function: ViewSyntaxHighlighting()

This command can be used to toggle on/off the display of Syntax Highlighting on files which are eligible for such display. This command overrides the option on the Configure | Syntax Highlighting dialog that enables and disables syntax highlighting for an individual programming language.

To create a temporary association between a file and a syntax highlighting language, or to disable syntax highlighting for a single file, use the View | Syntax Highlight As command.

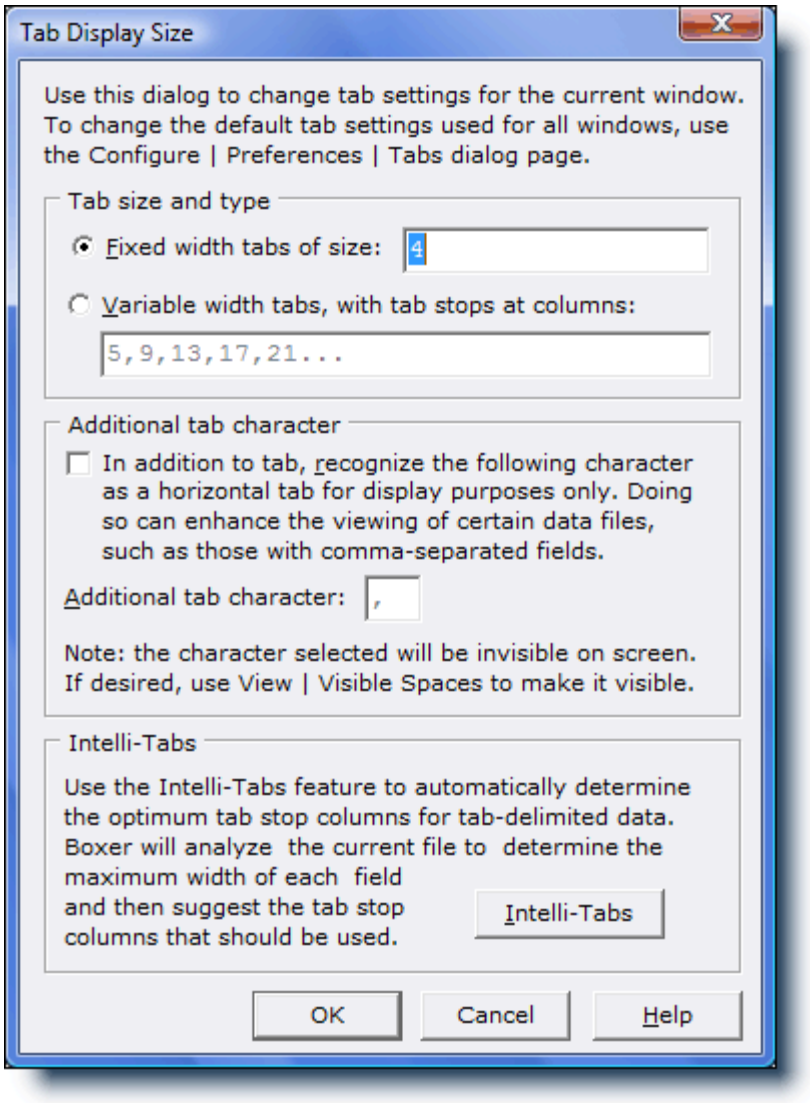
5.271 Tab Display Size

Menu: View > Tab Display Size

Default Shortcut Key: Alt+F9

Macro function: TabDisplaySize()

The Tab Display Size command is used to set the display width of the tab character within the current editor window. To set the default tab size for all future editing sessions, see the Configure | Preferences | Tabs dialog page.



Boxer supports the use of either fixed or variable width tabs. When fixed width tabs are used, the display width of a tab is a constant value, though the *effect* of a tab within text will depend on its column location. When variable width tabs are used, the display value of a tab is computed so as to cause a jump to the next tab stop. Variable width

tabs are sometimes referred to as *typewriter style tabs*, since they mimic the function of tab stops on early typewriters.

Tab size and type

Fixed width tabs of size *n*
Use this option to set the display size for fixed width tabs in the current file.

Variable with tabs, with tab stops at columns...
Use this option to designate the columns at which variable width tab stops should occur in the current file.

Additional tab character

When viewing certain data files, it can sometimes be helpful to treat an additional character as though it were a tab, for display purposes only. For example, when viewing a file containing comma-separated fields (CSV), designating a comma as the additional tab character will allow Boxer to display the file with each field in its own column, greatly enhancing its on-screen readability. This option would typically be used when variable width tabs are in use, and most likely in conjunction with the Intelli-Tabs feature described below.

When an additional character is designated as a tab, it will become invisible on screen, just as is the tab character. You can use the Visible Spaces command to make real tabs--and the additional tab character--visible on screen.


If a comma is designated as the additional tab character, please note that it is not possible to properly process quote and comma-delimited data files whose field data contains commas within the quoted fields. Boxer requires that the field separator character appear only between fields, and not within the data itself. In the help topic for the Replace command, the *Process $1, $2, $3... substring directives in the replace string* section contains an example that shows how embedded commas can be removed.


When an additional tab character is in use, the Tabs to Spaces command will treat that character as though it was a tab. This makes it possible to convert a file that uses a character-separated field format (CSV, for example), to a fixed width field format. See the topic Converting CSV Data to Fixed Width Format.

Intelli-Tabs

Use the Intelli-Tabs feature to automatically determine the optimum tab stop columns for tab-delimited data. Boxer will analyze the current file to determine the maximum width of each data field and then suggest the tab stop columns that should be used for optimum viewing.

The Intelli-Tabs feature can also be used on files containing fixed width field data. If tabs (or *'additional tabs'*, see above) are not found in the data, fixed width field data is assumed. Then a secondary analysis is made to try to determine the boundaries of the fields. If a range of lines is selected, the secondary analysis will be restricted to the selected range.

During its analysis, the Intelli-Tabs feature will often detect records whose field count differs from those of other records. When this happens, a report will be given, and the (first) non-conforming line number will be reported. As such, Intelli-Tabs can double as a useful tool for validating data files.

Boxer's default fixed width Tab Display Size is 4, which permits program source code with several indent levels to be displayed without exceeding the screen width. Many other programs, and most printers, will treat Tabs as having a display size of 8. You may need to make adjustments in order to print or display files with another program which does not use a Tab display size of 4. One remedy could be to use the Tabs to Spaces command to convert a copy of the file before using it with the other program. Note that Boxer's Print command will automatically convert Tabs to Spaces before sending its data to the printer, so there will be no such difficulty when printing files from within Boxer.

See the Insert Tab command for additional information about tabs.

After selecting the proper tab stops for optimum viewing, consider using the View | Shaded Tab Zones command to colorize the background of adjacent fields.

Tab settings can be designated on the command line using the -T option flag. See Command Line Options for more information.

Files that have Syntax Highlighting applied are eligible to have their tab stop settings defined as part of the syntax information for the language being highlighted. See Configure | Syntax Highlighting for more information. The parameter of interest is the *Tab Stops* parameter.

Tabs, spaces and newline characters can be made visible with the Visible Spaces command.

5.272 Tabs to Spaces

Menu: Block > Convert Other > Tabs to Spaces

Default Shortcut Key: none

Macro function: TabsToSpaces()

The Tabs to Spaces command can be used to convert the Tabs within a selected area of text into an equivalent number of Spaces. In doing so, Boxer uses the current display value of a Tab (View | Tab Display Size) to determine how many Spaces should be used.

No attempt is made to determine whether the Tabs being changed reside within a quoted string. Programmers should be careful when using this command, as changing Tabs within a quoted string my yield undesirable results if the string was to be displayed in a message on-screen.

The Spaces to Tabs and Tabs to Spaces commands are not opposites. If the Tab Display Size is 4, the sentence:

```
The<tab>dog<tab>ran<tab>wild.
```

would be converted by the Tabs to Spaces command to:

```
The<space>dog<space>ran<space>wild.
```

Running the Spaces to Tabs command on this sentence would not yield the original sentence. A sequence of two or more spaces is required before a tab character is considered for placement into the converted text.

If the Tab Display Size command has been used to designate an additional tab character for display purposes, the Tabs to Spaces command will treat that character as a tab when performing its conversion to spaces. This makes it possible to convert a data file that uses a character-separated field format into a fixed width field format. See Converting CSV Data to Fixed Width for more details.

5.273 Technical Support

Menu: Help > Technical Support

Default Shortcut Key: none

There are several ways to receive technical support for Boxer. The first and most obvious resource is the online Help. Online help contains detailed information on the configuration and use of Boxer, and for all of its commands. If you are having problems, please consult the relevant section of help before contacting us for support. You may also be able to find answers to some common questions on our website:

www.boxersoftware.com

Email
You can send electronic mail to us via the Internet. We prefer this method of support since it allows us to fully research a problem before responding. Also, we can sometimes reuse an earlier reply for a problem which has been experienced by more than one person. We typically check email several times a day:

support@boxersoftware.com

Telephone
You can also reach us by telephone Monday through Friday, 10:00 AM to 4:00 PM, Mountain Standard Time.

```
Voice: +1-602-485-1635
```

Postal Mail or Fax
Finally, you can mail or fax your inquiry to us. If you choose one of these methods,

please be sure to describe your problem fully and include any information which may help us to diagnose the problem. Whenever possible, please provide an email address so that we can make return contact quickly and easily.

Fax: +1-602-485-1636

Boxer Software
PO Box 14545
Scottsdale, AZ
85267-4545 U.S.A.

5.274 Templates (Configure)

Menu: Configure > Templates

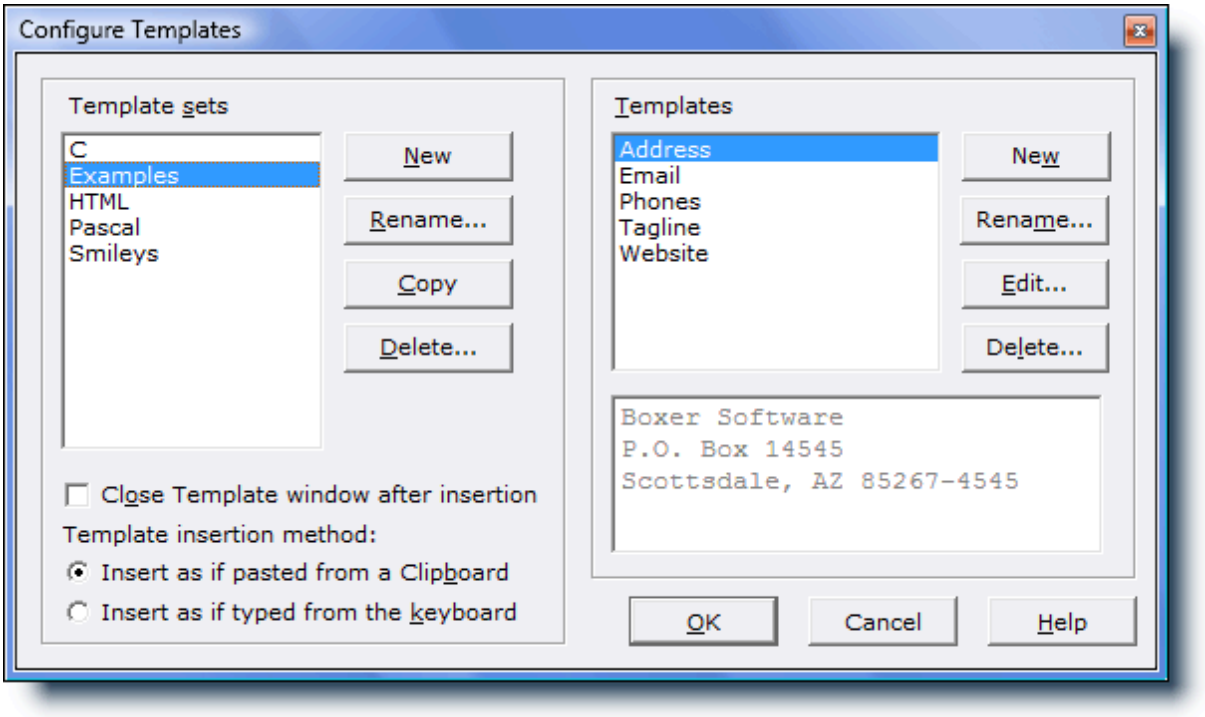
Default Shortcut Key: none

Macro function: ConfigureTemplates()

The Configure Templates command is used to create or edit *Templates*: user-defined text blocks which can be inserted into any text file from a pop-up selection menu. Once Templates have been defined, they can be inserted with the Templates command.

Templates are often used by programmers for defining the control structures of a programming language so that they can later be entered more quickly and without the chance of a typing error. The use of Templates, however, can be extended to facilitate the entry of *any* text, such as address blocks, copyright notices, phone numbers, part numbers, etc.

Template Sets and Templates are defined using the following dialog box:



Templates may consist of one or more lines of text with any formatting or indenting you choose. In addition, the template can dictate the placement of the text cursor, or how selected text should be used:

Text Cursor Placement
The Vertical Rule character (|) can be placed within a Template to dictate where the text cursor should be placed within the Template following its insertion. This allows, for example, programming code blocks to be defined in which the text cursor is placed between a pair of parentheses, ready for additional code to be typed.

Operating on Selected Text
The caret or circumflex character (^) can be placed within a Template to indicate that the template should operate on a text selection. A pair of examples will help to illustrate the power of this feature:

Example 1:
The template `<B>^</B>|` would cause the current text selection to be surrounded with HTML bold tags. The text cursor would be placed at the right of the closing bold tag.

Example 2:
The following Template:

```
<html>
<head>
|
</head>
<body>
^
</body>
```

```
</html>
```

could be run after using the Select All Text command to select the entire file. The effect would be to add the required HTML tags that help make an ordinary text file ready for viewing on the Internet. The text cursor would be place between the <head> and </head> tags, awaiting a title for the document.

Unindenting within a Template
If you need to unindent within a defined Template, use the tilde character (~) to designate the point at which the Backspace command should occur. The tilde will not be recognized in this way unless the *Insert as if typed from the keyboard* option is in force (see below).

Template Sets

A *Template Set* is a collection of Templates. Up to 100 Template Sets can be defined. Up to 500 Templates can be defined within any Template Set.

New
Use the *New* button to define a new Template Set. A pop-up dialog will appear into which the name of the Template Set is entered.

Rename
Use the *Rename* button to change the name of an existing Template Set.

Copy
Use the *Copy* button to make a copy of the currently selected Template Set.

Delete
Use the *Delete* button to delete the currently selected Template Set. A confirmation is required before the deletion occurs. If a Template Set is accidentally deleted, you can recover it by using the *Cancel* button.

Close Template window after insertion
Use this option to dictate whether or not the Template window should be closed after a Template is inserted into the edited text. Note that this option is maintained separately for each Template Set, permitting a different behavior to be defined as needed for different Template Sets.

Insert as if pasted from a Clipboard
Use this option if you prefer that Templates from the current Template Set be inserted into the text stream as if they had been pasted from a Clipboard. When this option is selected, the Autoindent setting will not influence the indent level of template text. Note that this option is maintained separately for each Template Set, permitting a different behavior to be defined as needed for different Template Sets.

Insert as if typed from the keyboard
Use this option if you prefer that Templates from the current Template Set be inserted into the text stream as if they had been typed from the keyboard. When this option is selected, the Autoindent setting will influence the indent level of template text, if the Template is inserted on an indented line. Note that this option is maintained separately

for each Template Set, permitting a different behavior to be defined as needed for different Template Sets.

If you need to unindent within a defined Template, use the tilde character (~) to designate the point at which the Backspace command should occur. The tilde will not be recognized in this way unless the *Insert as if typed from the keyboard* option is in force.

Templates

New
Use the *New* button to define a new Template. First, a dialog box will be presented to get the name of the new Template. Then an editing window will appear into which the Template text can be typed.

Rename
Use the *Rename* button to change the name of an existing Template.

Edit
Use the *Edit* button to edit the content of an existing Template.

Delete
Use the *Delete* button to delete the currently selected Template. A confirmation is required before the deletion occurs. If a Template is accidentally deleted, you can recover it by using the *Cancel* button.


Boxer's Template information is stored in the file `Template.ini`, and its format is that of a simple text file, not a binary file.


If you need to insert one of the special characters (| or ^) into a template in its textual form, use either || or ^^. If you need to insert the special character ~ into a template, use \~.


If the need arises to insert a single character which is not easily typed from the keyboard, consider using the Insert Symbols feature rather than defining a single character Template. The Insert Symbols feature permits a defined character to be entered using a single keystroke.

5.275 Templates (Insert)

Menu: Tools > Templates

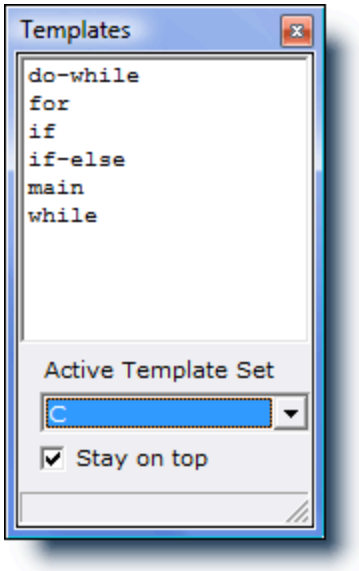
Default Shortcut Key: Ctrl+T

Macro function: Templates()

The Templates command is used to select a Template for insertion at the text cursor. Boxer's Templates are an excellent way to save and recall small pieces of text such as address blocks, copyright notices, programming language constructs, email addresses

and any other text block which is used frequently during editing.

When the Insert Template command is issued, the Templates menu appears in a popup window:



The active Template Set is displayed at the bottom of the window, and the Templates within that set are displayed in the main listing area. Press the first letter of the Template name or cursor with the arrow keys to select the desired Template. Press *Enter* (or double click with the mouse) to insert the selected Template into the file.

Right-clicking on a selected item summons the Template context menu. The context menu provides options to insert the selected item, or to copy it to the current clipboard.

Template Sets and Templates are defined using the Configure | Templates command.

Text Cursor Placement
The Vertical Rule character (|) can be placed within a Template to dictate where the text cursor should be placed within the Template following its insertion. This allows, for example, programming code blocks to be defined in which the text cursor is placed between a pair of parentheses, ready for additional code to be typed.

Operating on Selected Text
The caret or circumflex character (^) can be placed within a Template to indicate that the template should operate on a text selection. A pair of examples will help to illustrate the power of this feature:

Example 1:
The template <B>^</B>| would cause the current text selection to be surrounded with HTML bold tags. The text cursor would be placed at the right of the closing bold tag.

Example 2:
The following Template:

```
<html>
<head>
|
</head>
<body>
^
</body>
</html>
```

could be run after using the Select All Text command to select the entire file. The effect would be to add the required HTML tags that help make an ordinary text file ready for viewing on the Internet. The text cursor would be place between the <head> and </head> tags, awaiting a title for the document.

Unindenting within a Template
If you need to unindent within a defined Template, use the tilde character (~) to designate the point at which the Backspace command should occur. The tilde will not be recognized in this way unless the *Insert as if typed from the keyboard* option is in force (see Configure | Templates).

If you need to insert one of the special characters (~, | or ^) into a template in its textual form, use either ~~, || or ^^.

To ensure that special characters are displayed in the Template window as they will appear when inserted into the editor, the Template window uses the same font as is used in the editor itself.

If the need arises to insert a single character which is not easily typed from the keyboard, consider using the Insert Symbols feature rather than defining a single character Template. The Insert Symbols feature permits a defined character to be entered using a single keystroke.

If the Template window is left on-screen when Boxer is closed, it will be automatically reopened if the edit session is later restored.

5.276 Text Highlighting (Configure)

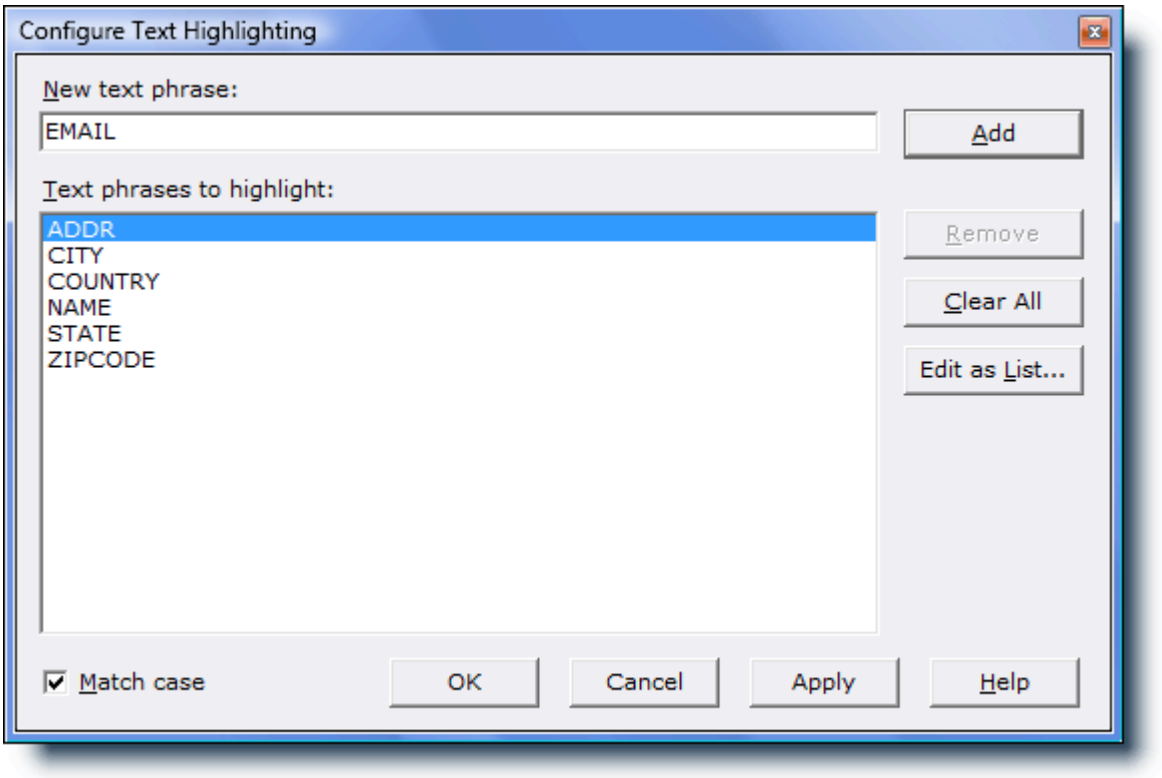
Menu: Configure > Text Highlighting

Default Shortcut Key: none

Macro function: ConfigureTextHighlighting()

The Configure Text Highlighting command allows the user to designate any number of text strings for on-screen highlighting. This feature might be used to make table headings stand out, or to add emphasis to any class of words or phrases that might be desired. The highlighting strings are saved and restored from session to session. The color used to highlight the designated strings is configurable on the Configure | Colors dialog. Text Highlighting can be applied to normal text files, or to program files which

are already being Syntax Highlighted. The highlighting of strings can be quickly toggled on/off by using the View | Text Highlighting command.



The Find command has an option to highlight all matches of a given search string.

The Apply Highlighting command can be used to quickly add text to the list of phrases to be highlighted.

5.277 Text Highlighting (View)

Menu: View > Text Highlighting

Default Shortcut Key: Alt+F8

Macro function: ViewTextHighlighting()

This command is used to toggle on/off the text highlighting performed by either the Text Highlighting command, or the *Highlight all matches* feature of the Find command.

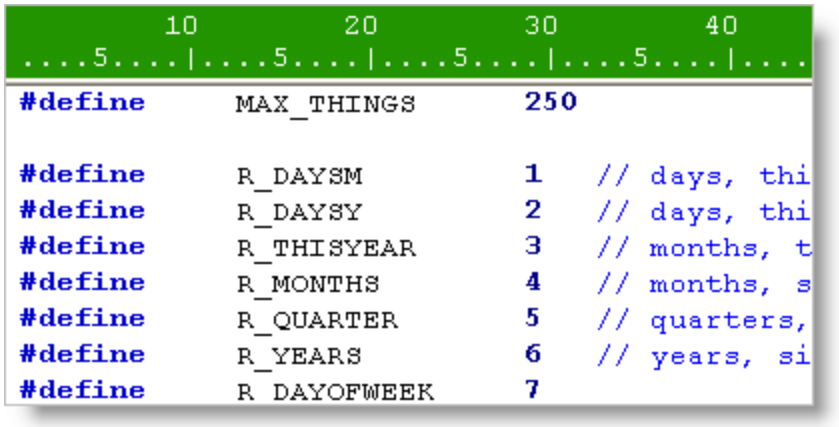
5.278 Text Ruler

Menu: View > Text Ruler

Default Shortcut Key: Alt+F5

Macro function: ViewTextRuler()

The View Text Ruler command is used to toggle on or off the horizontal ruler at the top of the editing window.



The Text Ruler labels the column numbers of the file being displayed. When the view of the file is scrolled to the right, the ruler values scroll along with the file. Clicking on a column number within the ruler will move the text cursor to that column on the current line. Clicking at the far right of the Ruler will cause the file to scroll to the right.

The current column number is also displayed in the Status Bar.

To enable the display of *line* numbers, use the View Line Numbers command.

Clicking on the Ruler with the right mouse button provides access to its context menu. The menu has an option to turn off display of the Ruler.

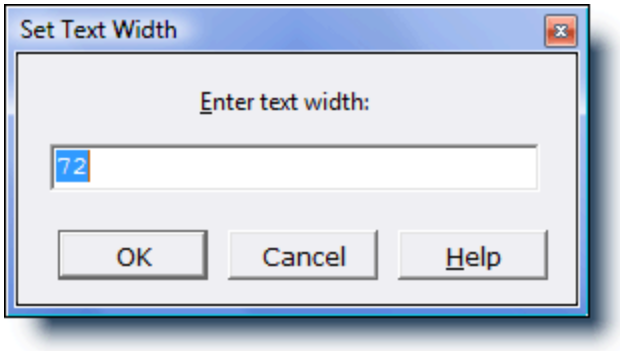
5.279 Text Width

Menu: Paragraph > Text Width

Default Shortcut Key: Ctrl+W

Macro function: TextWidth()

The Text Width command is used to set the column at which text justification commands will wrap words to the next line. A popup dialog box will appear to retrieve a new value for the Text Width:



The Text Width value is used by the following commands during their operation:

Typing Wrap
Reformat
Quote and Reformat
Align Left
Align Center
Align Right
Align Smooth

The current Text Width is displayed on the Status Bar, next to the 'w' indicator which displays Typing Wrap status. The Text Width command can also be issued by double clicking on the Text Width value in the Status Bar.

The maximum value for the Text Width command is 9999.

5.280 Tile Across

Menu: Window > Tile Across

Default Shortcut Key: none

Macro function: TileAcross()

The Tile Across command can be used to resize and reposition all editor windows such that the *client area* is fully occupied, and the windows are arranged left-to-right.

Minimized windows are not affected by this operation.

If the number of open windows will not permit a left-to-right arrangement, Windows will use an alternative arrangement which allows all windows to fit.

5.281 Tile Down

Menu: Window > Tile Down

Default Shortcut Key: none

Macro function: TileDown()

The Tile Down command can be used to resize and reposition all editor windows such that the *client area* is fully occupied and the windows are arranged top-to-bottom.

Minimized windows are not affected by this operation.

If the number of open windows will not permit a top-to-bottom arrangement, Windows will use an alternative arrangement which allows all windows to fit.

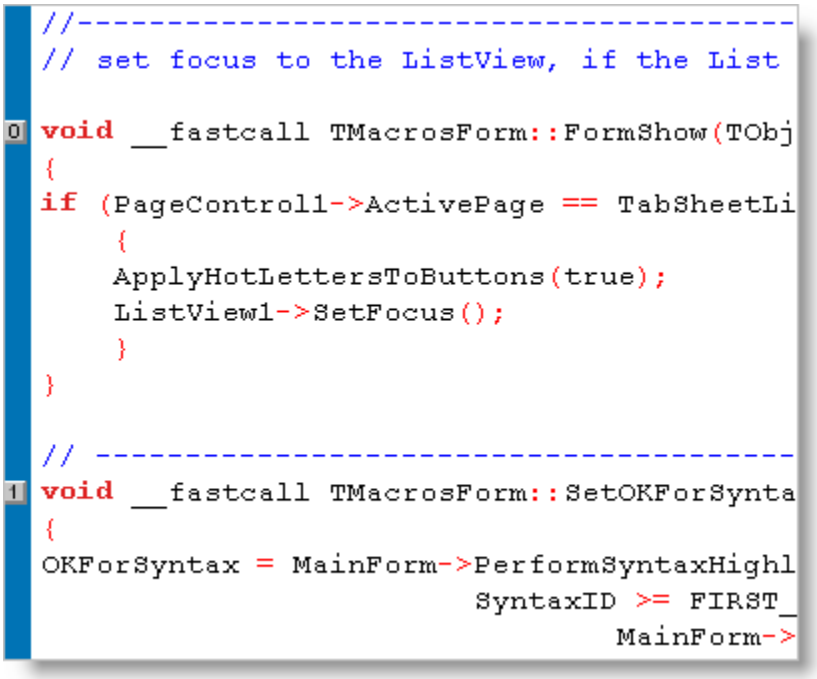
5.282 Toggle Bookmark

Menu: Jump > Toggle Bookmark

Default Shortcut Key: F9

Macro function: ToggleBookmark()

The Toggle Bookmark command places a bookmark at the current location of the text cursor, or clears a bookmark if the line is already bookmarked. Bookmarks are displayed at the far left edge of the window as a small number (0-9) within a gray box. Up to ten bookmarks can be placed in a file at any one time.



If the bookmarked column is altered due to the addition or deletion of text on the bookmarked line, the bookmark will be adjusted automatically. If a line containing a

bookmark is deleted, the bookmark will be cleared.

Bookmarks can be used to mark various points of interest within a text file. Once one or more lines have been bookmarked, you can use the Previous Bookmark and Next Bookmark commands to move among the bookmarked lines. The Bookmark Manager can be used to view all bookmarked lines in a single view, and navigate to, or delete, selected bookmarks.

Bookmarks will persist for the current editing session, and will be restored when restoring an edit session.

The display of bookmarks is controlled by the View | Bookmarks command. Bookmarks remain operational even if they are not currently being shown on-screen.

5.283 Toggle Read-Only

Menu: File > Toggle Read-Only

Default Shortcut Key: None

Macro function: ToggleReadOnly()

The Toggle Read-Only command can be used to toggle the status of the current file between read-only and writable. The current state is displayed in the status bar. 'WR' denotes that the file is writable. 'RO' indicates that the file is read-only.

This command duplicates the functionality available by double-clicking within the status bar in the read-only display panel, but it also provides the ability to assign that function to a key sequence (via Configure | Keyboard), if desired.

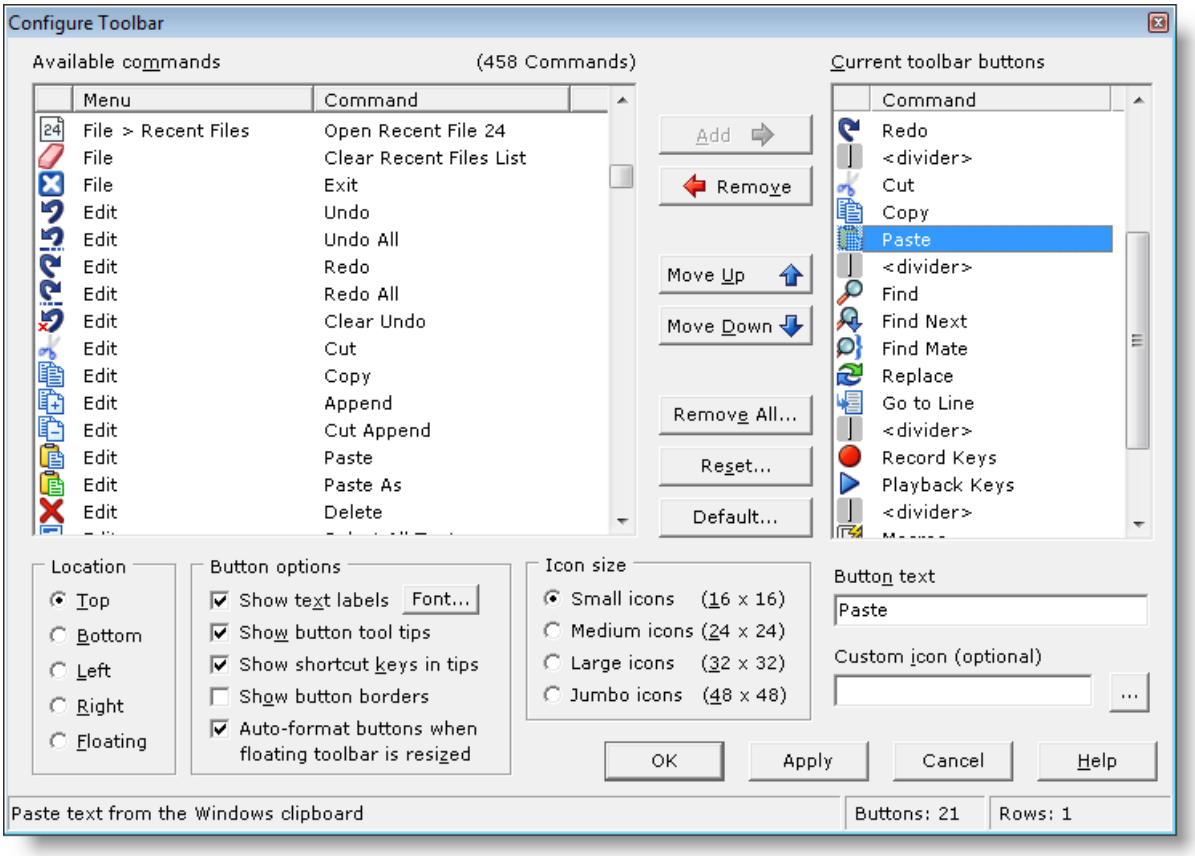
5.284 Toolbar (Configure)

Menu: Configure > Toolbar

Default Shortcut Key: none

Macro function: ConfigureToolbar()

The Configure Toolbar command can be used to add or delete buttons from the toolbar, and to change the relative position of those buttons. Options are also available to control the location of the toolbar, whether the buttons are to have text labels, and several other features related to the toolbar.



Available commands

The *Available commands* listbox contains entries for all editor commands that are eligible for placement on the toolbar. The commands are initially presented in menu-order, but can be sorted alphabetically by clicking on the *Command* header at the top of the list. Click on the *Menu* header to return the list to menu-order presentation.

To add a new command to the toolbar, select the command in the *Available commands* listbox and click the *Add* button.

If you first select a button in the *Current toolbar buttons* listbox, you'll be able to control where a new button is added. Buttons are added below the selected button in the *Current toolbar buttons* listbox.

In addition to the editor commands, several special entries can be used to control the appearance of the toolbar. The *half space* and *full space* entries can be used to insert extra spacing between toolbar buttons. The *divider* entry can be used to insert a visible divider. The *new row* entry can be placed on the toolbar to create an additional row. All toolbar buttons situated below a *new row* entry will appear on a new row of the toolbar.

Current toolbar buttons

The *Current toolbar buttons* listbox contains entries for all of the buttons that are currently on the toolbar. The order of the list controls the order the buttons will appear on the toolbar. Use the *Move Up* and *Move Down* buttons to move a selected button within the list. To remove a button from the toolbar, select the button and click the *Remove* button. The *Reset* button can be used to restore the toolbar to its most recent configuration. The *Default* button can be used to restore the toolbar to its default configuration.

Button text
This edit box provides control over the text that will appear below a toolbar button, when that option has been selected. By default, the full name of the command will be used, but this text can be changed if desired.

Custom icon
This option allows a user-defined icon to be used in place of Boxer's default icon for a given command. Some users might wish to customize Boxer's look by using special icons, or assign more meaningful icons to extra commands that have been added to the toolbar. For example: the Run Macro ## commands all use the same toolbar icon and, unless new icons were used, would differ only in the text that is assigned to each button.

Custom icons can be supplied in either icon (.ICO) or bitmap (.BMP) format. For best results, use an image that matches the size of the icons that are in use. If the custom image does not match the icon size that's in use, the image will be scaled automatically. Boxer uses the convention that the color of the pixel in lower left corner of the image defines the background color. Pixels of this color are treated as invisible, allowing the image to blend more naturally with the background color of the button on which it is drawn.

Location

Use the *Top*, *Left*, *Bottom*, *Right* and *Floating* radio buttons to control the location of the toolbar. The location can be easily changed later on by right-clicking on the toolbar, or by dragging it from its current location.

Button options

Show text labels
Use this option to control whether or not text labels will appear below each toolbar button. The *Font* button allows the font and size to be selected from the available fonts on your system.

Show button tool tips
Use this option to control whether or not *tool tips* will be displayed when the mouse is allowed to hover atop a toolbar button.

Show shortcut keys in tips
Use this option to control whether or not a shortcut key will appear within the toolbar button tool tips.

Show button borders
Use this option to place a visible border around each toolbar button.

Auto-format buttons when floating toolbar is resized
Use this option to control the formatting of the toolbar when it is displayed in floating mode. If auto-format is selected, the toolbar will ignore any *new row* entries within the toolbar and format itself to fill the size of the toolbar. If auto-format is not selected, a floating toolbar will wrap to a new row only when a *new row* entry occurs.

As you experiment with the various toolbar button options, use the *Apply* button to preview the toolbar as it is currently configured.

Icon size

Small / Medium / Large / Jumbo
Use these options to control the size of the icons displayed on the toolbar. The natural size of Boxer's built-in icons is 16 x 16; other icon sizes are achieved by scaling these icons accordingly. You'll find that the built-in 16 x 16 icons scale smoothly to 32 x 32 and 48 x 48. The 24 x 24 icon size will show imprecision for some icons, and may be most useful when a set of custom 24 x 24 icons is being assigned.

A small collection of icons have been supplied for inspiration and experimentation. These icons are from the public domain icon set created by the Tango Desktop Project.

The *Jumbo icons* setting is very large, and may be useful to visually impaired users.

Due to internal limitations, the largest image that can be loaded from an icon (.ICO) file is 32 x 32. If you are using the *Jumbo icons* option, and you don't want your image to be scaled from 32 x 32 to 48 x 48, use a 48 x 48 bitmap (.BMP) file as the source image file.

5.285 Toolbar (View)

Menu: View > Toolbar

Default Shortcut Key: none

Macro function: ViewToolbar()

The View Toolbar command is used to toggle on and off the display of Boxer's Toolbar. The Toolbar is located just below the main menu bar, and looks like this:

The Toolbar provides one-click access to Boxer's most common commands. When the mouse cursor is allowed to hover over a Toolbar button, a *Tool Tip* will popup showing the name of the command associated with that button. The display of tool tips can be disabled with an option on the Configure | Preferences | Display options page. The option is titled *Display Toolbar button tool tips*. There is also an option provided to

display shortcut keys within the tool tips.

The Toolbar can be repositioned to any edge of the window by clicking on an open area in the Toolbar and dragging it to the new location. The Toolbar can also be repositioned--or turned off--by right-clicking on it to gain access to its context menu.

Toolbar buttons can be displayed in a raised, 3-D style using an option on the Configure | Preferences | Display options page. The option is titled *Display Toolbar buttons in 3-D style*.

☞ The dollar bill icon which appears at the far right of the Toolbar is used to summon Boxer's Order Form. This icon is present only in the evaluation version of Boxer.

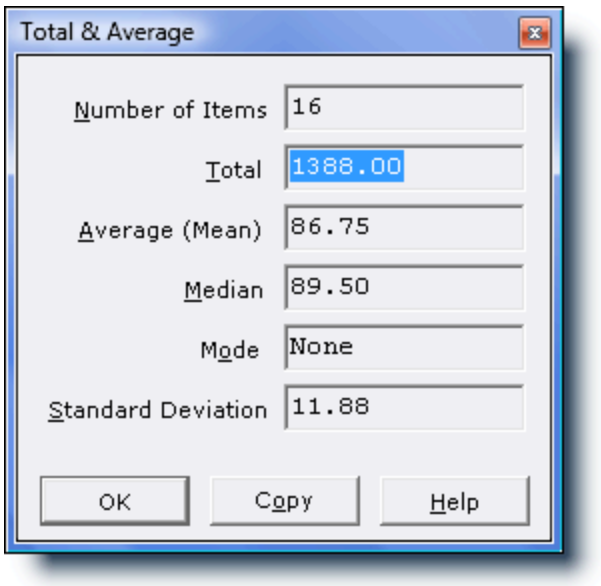
5.286 Total and Average

Menu: Block > Total and Average

Default Shortcut Key: None

Macro function: TotalAndAverage()

The Total and Average command can be used to obtain a report on the numeric data contained on a range of selected lines. In addition to computing the total and average (*mean*), the *number of items*, *median*, *mode* and *standard deviation* are also reported.



The *median* is the midpoint between the low and high values in the data.

The *mode* is the most frequently occurring value among the data.

The *standard deviation* provides a measure of the dispersion of the values within the data.

The Total and Average command can be used on ranges of numbers that include thousands separators (commas or periods, typically). The data is analyzed to determine what convention for the thousands and decimal separator is in use. The format of the report is adjusted to reflect the format in use.

Total and Average can be used on values that contain a leading currency symbol. The following symbols will be ignored during computation: dollar sign, yen sign, pound sterling sign, euro sign.

The Total and Average command reports its results using read-only edit boxes so that any of the fields can be copied to the Windows clipboard. The *Copy* button can be used to copy the full report to the current clipboard.

5.287 Typing Wrap

Menu: Paragraph > Typing Wrap

Default Shortcut Key: Ctrl+F5

Macro function: TypingWrap()

The Typing Wrap command is used to toggle on and off typing wrap mode. When Typing Wrap mode is on, text typed from the keyboard will be wrapped to the next line when the Text Width value is exceeded. When Typing Wrap mode is off, typed text will not be wrapped to the next line until the *Enter* key is pressed.

When Boxer wraps text to the next line in Typing Wrap mode, it does so by adding a true newline (hard line ender) character at the end of the line. To wrap text visually, without introducing a hard line ender into the file, see the Visual Wrap command.

Typing Wrap mode is maintained separately for each edited file. Activating Typing Wrap in one file does not affect the Typing Wrap mode for other edited files.

The current file's Typing Wrap mode is displayed on the Status Bar. An uppercase 'W' indicates that Typing Wrap is on. A lowercase 'w' indicates that Typing Wrap is off. The Typing Wrap command can also be issued by double clicking on the 'w' value in the Status Bar.

See also the Visual Wrap command.

Typing Wrap will break lines between HTML or XML tags, when appropriate, even if an intervening space is not present.

Prior to Boxer v14, the Typing Wrap command was called Word Wrap. When the Visual Wrap command was added, the command was changed to Typing Wrap for clarity.

5.288 Uncomment

Menu: Block > Uncomment

Default Shortcut Key: Shift+F5

Macro function: Uncomment()

The Uncomment command will remove commenting from the current line--or the selected text--according to Boxer's syntax information about the language being edited (see Configure | Syntax Highlighting).

The Comment command can be used to apply commenting to the current line or to selected text.

5.289 Undo

Menu: Edit > Undo

Default Shortcut Key: Ctrl+Z

Macro function: Undo()

The Undo command can be used to reverse the effect of the most recent change to the current file. Successive Undo commands will have the effect of stepping back in time, with each command undoing the previous change, until the limits of Undo become exhausted. If a change is undone which you'd like to get back, the Redo command can be used to 'undo undo'.

By default, cursor motion changes will also be undone. For example, if you're editing mid-file and then jump to start of file to check something, Undo can be used to return you to your previous location. If you prefer that cursor motion commands *not* be stored for Undo, uncheck the relevant option on the Configure | Preferences | Editing 1 dialog page.

The Undo command does not affect the content of the current clipboard.

The size of the Undo buffer (in bytes) can be controlled on the Configure | Preferences | Editing 1 options page. The option is titled *Undo buffer size*. Values between 2048 and 65535 may be entered. The default value is 65535, which is also the maximum. There is little reason to select smaller values, as the memory cost is small compared to the utility that Undo provides.

Undo information is stored separately for each file, so there is no chance that excessive editing within one file can exhaust the undo capacity in another file.

An option is also provided to control whether or not Undo is allowed after the Save command. This option is titled *Allow Undo after File Save*

5.290 Undo All

Menu: Edit > Undo All

Default Shortcut Key: none

Macro function: UndoAll()

The Undo All command can be used to reverse the effect of all changes for which undo information is available. Unless you feel sure about the number of changes which have been recorded by Undo, it is often safer to use the Undo command to step singly through the changes so that their effect can be seen on-screen before proceeding. Should the Undo All command go 'too far', the Redo All command can be used to reverse its effect, or the Redo command can be used to restore the changes one at a time.

5.291 Undo All Closed Tabs

Menu: View > File Tabs > Undo All Closed Tabs

Default Shortcut Key: none

Macro function: UndoAllClosedTabs()

The Undo All Closed Tabs command can be used to reopen all files that have been closed during the current editing session. The names of the last ten (10) files are stored for reopening.


The "Closed Tabs List" at the bottom of the View | File Tabs submenu shows the names of the files that are eligible to be reopened, and allows files within the list to be selectively reopened.

5.292 Undo Closed Tab

Menu: View > File Tabs > Undo Close Tab

Default Shortcut Key: none

Macro function: UndoCloseTab()

The Undo Close Tab command can be used to reopen the file that was last closed during the current editing session. This command makes it easy to reopen a file if it was closed accidentally.


The "Closed Tabs List" at the bottom of the View | File Tabs submenu shows the names of the files that are eligible to be reopened, and allows files within the list to be selectively reopened.

Clicking the middle mouse button in an open area of the file tab bar is taken as a shortcut gesture to reopen the last closed file tab.

5.293 Unformat

Menu: Paragraph > Unformat

Default Shortcut Key: Ctrl+Alt+F10

Macro function: Unformat()

The Unformat command can be used to convert the lines of the current paragraph into a single, long line. The Unformat operation begins on the current line and includes all lines to the end of the current paragraph. The text cursor is advanced to the first line of the next paragraph following Unformat, so that successive Unformat commands will move smoothly through the document.

If a range of lines is selected, all paragraphs within the selected range will be processed. Use the Select All Text command before Unformat to process an entire file, but first check to be sure that the file doesn't contain tables or lists which would be adversely affected by the new formatting.

See also the Soften Line Enders command.

If the total length of the paragraph being unformatted is greater than the maximum line length (see Sizes and Limits), the operation will use multiple lines.

The Unformat command is useful for preparing text that is to be imported into a word processor, email client or other programs that perform 'soft formatting'. Programs such as these sometimes require that extra newlines be removed before imported text can be properly formatted.

5.294 Unformat XML / XHTML

Menu: Tools > Unformat XML / XHTML

Default Shortcut Key: none

Macro function: UnformatXML()

The Unformat XML / XHTML command can be used to remove formatting (newline and indentation) from an XML or XHTML file. This command can be used to remove formatting that was added by the Format XML / XHTML command, or to remove formatting from a file from some other source. After formatting is removed, the file will reside on one long line (subject to Boxer's limitation on line length).

By default, the unformat operation is applied to the whole file. If a range of lines is selected, unformat will be performed on the range of lines selected.

Please see the Format XML / XHTML command for a lengthy discussion of this topic.

5.295 Unhighlight Matches

Menu: Search > Unhighlight Matches

Default Shortcut Key: none

Macro function: UnhighlightMatches()

The Unhighlight Matches command removes on-screen highlighting from any text strings that matched the most recent Find operation.

The *Highlight all matches* option on the Find dialog causes matched strings to be highlighted throughout the current file. The Unhighlight Matches command removes that highlighting, without disabling the Find dialog option. When a new text string is searched for and found, it will again be highlighted. Put another way: this command provides a means to remove the highlighting added by the Find command without the need to disable the highlighting feature altogether, or perform a new, contrived search that is designed to fail.

5.296 Unindent

Menu: Block > Unindent

Default Shortcut Key: Shift+Backspace

Macro function: Unindent()

The Unindent command causes a selected range of lines to be unindented by one space or one tab character. If the range of lines to be unindented contains lines with varying levels of indent--or mixed indents of spaces and tabs--the Unindent command can still be used without concern. Once all of the *whitespace* on a given line has been deleted, Unindent will not remove additional characters, though it will continue to operate on other lines within the selection with remaining indent.

If text is not selected, Unindent will act upon the current line.

5.297 Unskip All

Menu: View > File Tabs > Unskip All

Default Shortcut Key: none

Macro function: none

The Skip command can be used to mark a file/window so that it will be skipped over by

Window Previous and Window Next when these commands are used to cycle through open files. The Unskip All command restores the state of and and all previously skipped windows to unskipped.


The File Tab context menu also includes options to toggle the skip state for the current file, or to set or clear the skip status for all open files.

5.298 Update All

Menu: Project > Update All

Default Shortcut Key: none

Macro function: ProjectUpdateAll()

Use the Update All command to update the active project file with the current editing options for *all* files within the project.

The project file stores the following information about the files contained in the project:

- window sizes and positions
- cursor location
- active file
- bookmarks
- tab stops
- typing wrap mode
- hex editing mode
- file tab arrangement


If you would like the project file to be updated automatically for all member files, use the Project | Auto-Update feature.

5.299 Update One

Menu: Project > Update One

Default Shortcut Key: none

Macro function: ProjectUpdateOne()

Use the Update One command to update the active project file with the current editing options for the active file.

The project file stores the following information about the files contained in the project:

- window sizes and positions
- cursor location
- active file
- bookmarks

- tab stops
- typing wrap mode
- hex editing mode
- file tab arrangement


If you would like the project file to be updated automatically for all member files, use the Project | Auto-Update feature.

5.300 User Lists

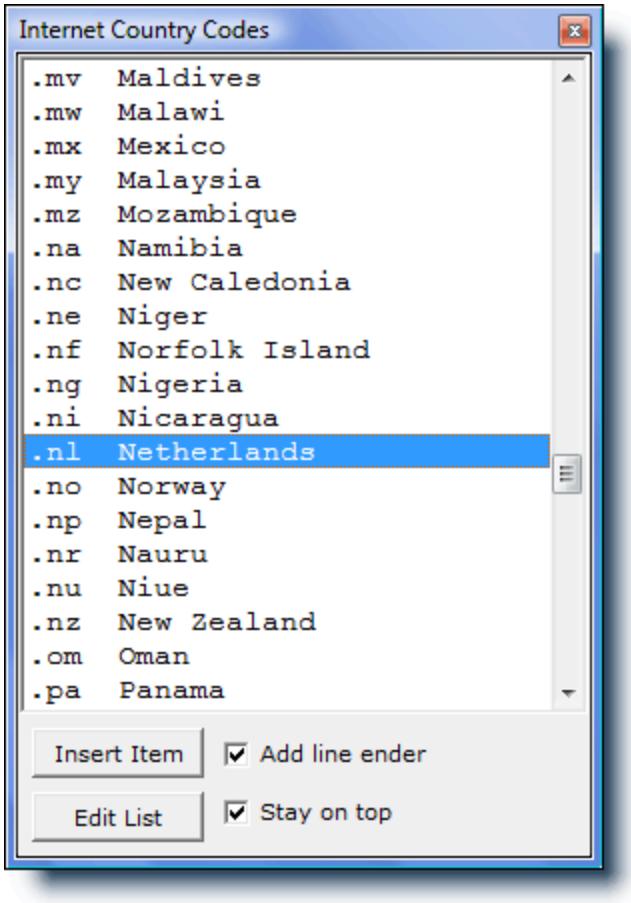
Menu: Tools > User Lists

Default Shortcut Key: none

Macro function: UserList()

The User Lists command provides access to a submenu of user-defined lists. Several example User Lists have been supplied with Boxer to suggest ways in which this feature might be used. You will no doubt think of many other ways.

Selecting a list from the User Lists submenu results in a popup window which displays the items in the list:



User Lists can be used for reference or to insert text strings into the file being edited. To insert a list item at the text cursor, double-click on the entry, highlight it and press *Enter*, or click the *Insert Item* button. If you would like a line ender to be added after inserting the item, use the *Add line ender* option.

Right-clicking on a selected item summons the User List context menu. The context menu provides options to insert the selected item, or to copy it to the current clipboard.

To advance quickly to an item in the list, enter its first letter from the keyboard.

To edit an existing list, click the *Edit List* button. To edit an empty list, simply select it from the User Lists submenu. In either case, the file which defines the list will be loaded into an editor window and can be edited in the usual way. The title of the list appears on the first line of the file and will be placed in the title bar of the popup window. The list items are placed one item per line, beginning on line two.

The maximum length of a User List item is 256 characters. The maximum length of a User List title is 40 characters.

The files which define User Lists are kept in Boxer's *data folder* in a subdirectory called 'User Lists' and are named `userlist.001`, `userlist.002`, etc.

If you prefer that the User List window remain atop other windows, select the *Stay on*

top option. The User List windows are *non-modal* windows, which allows them to remain on-screen after *focus* has been returned to another editing window. Multiple User List windows can be opened simultaneously.

Email and URL Addresses
If a User List entry contains either an email or internet address, Boxer will launch the default email client or internet browser when the entry is double-clicked. This makes it possible for a User List to be used to create a list of email contacts or favorite websites. Email addresses can be entered in any of the following formats.

```
Boxer Software <sales@boxersoftware.com>
<sales@boxersoftware.com>
sales@boxersoftware.com
```

Mailto extensions can be used within an email address. They are appended to the email address following a question mark (?). Here are some examples:

```
sales@boxersoftware.com?cc=joe@mycompany.com
sales@boxersoftware.com?bcc=bill@mycompany.com
sales@boxersoftware.com?subject=Order a Site License for Boxer
sales@boxersoftware.com?body=this text will appear in the
message body
```

Multiple mailto extensions can be combined with the ampersand (&) symbol:

```
sales@boxersoftware.com?subject=Order  Boxer&cc=joe@mycompany.com
```

Note: Mailto extensions are not supported by all email clients. Experiment with your email client to learn its capabilities.

Sin Since an address can be launched with a double-click, the Enter key retains the function of inserting the text of the entry into the current file.

 To ensure that special characters are displayed in the User List window as they will appear when inserted into the editor, the User List uses the same font as is used in the editor itself.

 If a User List is left on-screen when Boxer is closed, it will be automatically reopened if the edit session is later restored.

5.301 User Tools (Configure)

Menu: Configure > User Tools

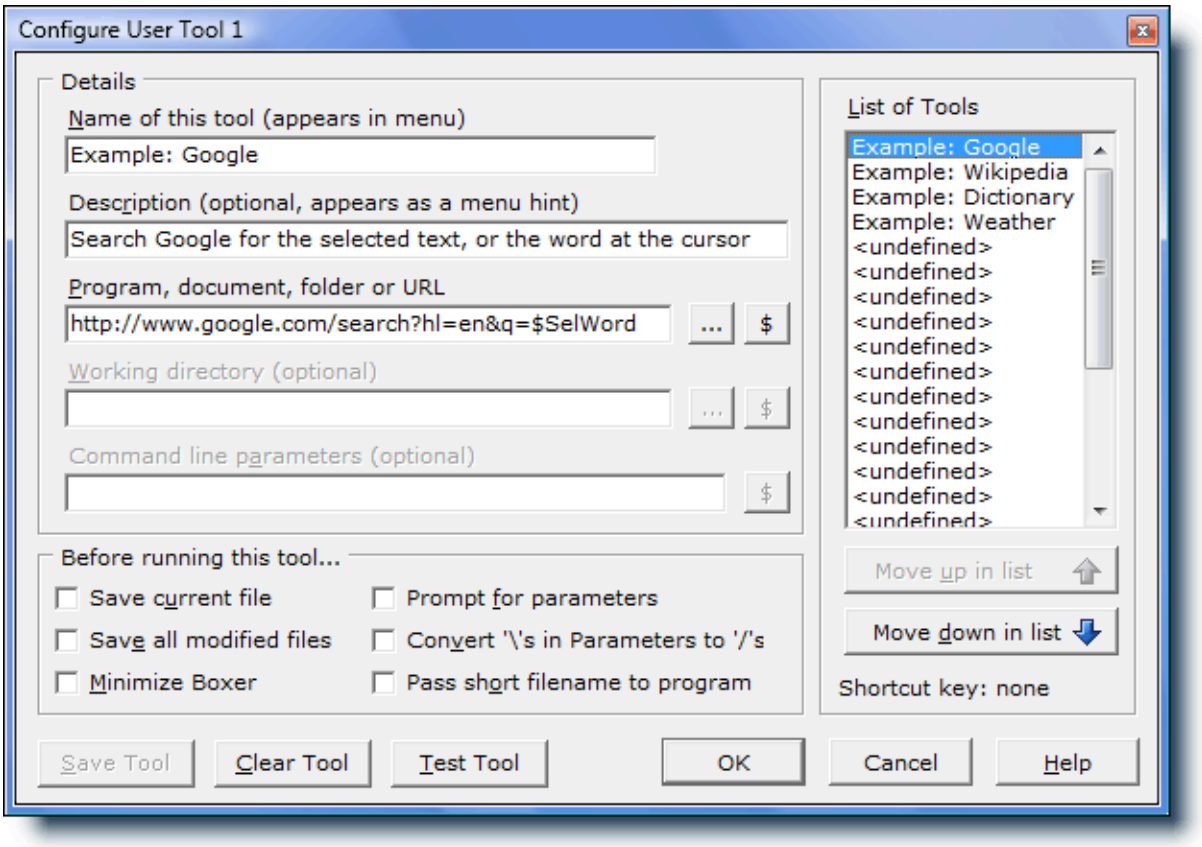
Default Shortcut Key: none

Macro function: ConfigureUserTools()

The Configure User Tools command is used to define and configure up to 24 external programs which can be run from the User Tools submenu on the Tools menu. A variety

of *Tools Macros* is available which makes it possible to control the information which is passed to the program being run.

A common use for a User Tool would be to send the name of the current file to an external program which processes the file in some way. Examples of such programs are assemblers, compilers, grammar checkers, parsers, etc.



Use of the Configure User Tools dialog box is described below:

Details

Name of this tool
Use this edit box to supply the name for the User Tool being defined. The name supplied will appear in the User Tools submenu when definition is complete. Up to 20 characters can be used.

Description
Use this edit box to supply an optional description for the tool being defined. This description will appear as a menu hint for the Tools | User Tools menu entry that corresponds to this tool.

Program, document, folder or URL
Use this edit box to supply the full filepath of the program which is to be run. The button with the ellipsis (. . .) can be used to browse for and select the desired program.

If the program selected has an associated icon, it will be displayed to the right of the *Name* edit box.

Tools Macros can be placed into the *Program* field by clicking on the '$' button. For example, you might use the $SelWord directive to pass the word at the cursor--or a short text selection--to a web-based resource that performs a search for that term. The URL:

```
http://www.google.com/search?hl=en&q=$SelWord
```

would cause the word at the cursor to be sent to Google for search results.


The $Sel and $SelWord directives are processed specially when used in a URL: any embedded spaces that might result from expansion are automatically converted to plus signs (+) to create a web-friendly URL.

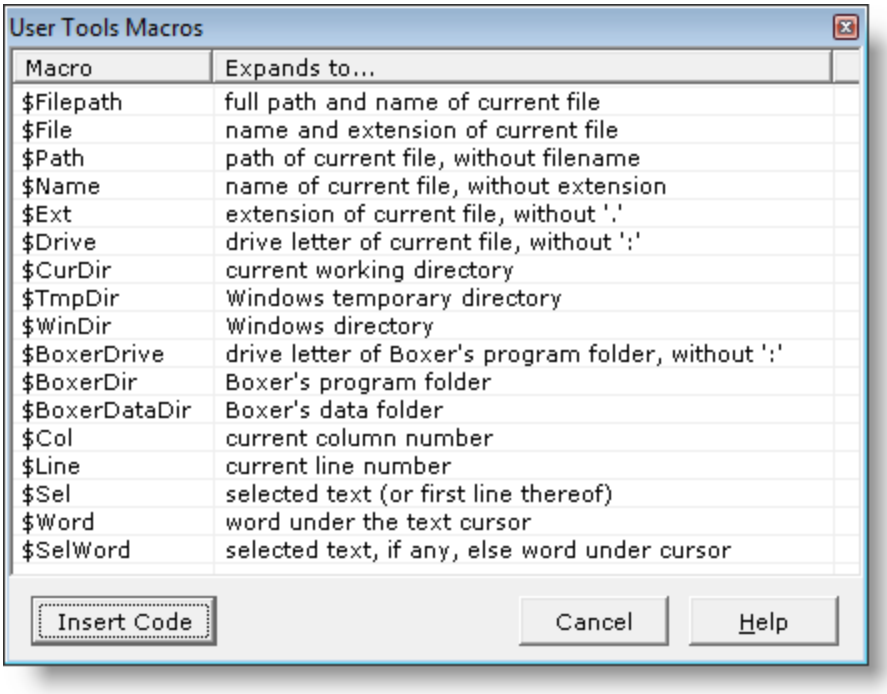
Working Directory
Use this edit box to supply the working directory for the program being defined. The working directory will become the current directory for the program being run. The button with the ellipsis (...) can be used to browse for and select the working directory.


The $Path Tools Macro (among others) can also be used in the *Working Directory* field as a means of specifying the working directory.

Command Line Parameters
The *Command Line Parameters* edit box is used to supply command line parameters to the program being defined. These parameters might be option flags required by the program or any other such information.

Boxer will recognize several different *Tools Macros* in the *Command Line Parameters* edit box to pass information to the defined program. When a macro appears in the *Command Line Parameters* field, it will be expanded to its equivalent text at the time the User Tool is run.

Clicking the ellipsis (...) button to the right of the edit box presents the *Tools Macros* list:



Macros are available to pass the current filepath, filename, extension and other portions of the filename. The line and column number can also be passed, as can the word beneath the text cursor. Selected text (or the first line of a multi-line selection) can also be passed. If needed, two or more macros can be placed in the *Command Line Parameters* edit box.

☞ If it is anticipated that the file and/or path being passed to the User Tool might contain an embedded space, be sure to enclose the $Filepath directive in double quotes to ensure proper handling.

Before running this program...

Save current file
Use this option to save any changes in the current file before running the defined User Tool. Be sure to use this option when defining a User Tool which will operate on the current file, so the program has access to the most recent changes you've made.

Save all modified files
Use this option to save any changes within all edited files before running the defined User Tool.

Minimize Boxer
Use this option to request that Boxer be minimized to the task bar while the User Tool is running.

Prompt for parameters
Use this option to request that Boxer prompt for parameters before running the defined program. This option is useful when running a program whose command line parameter(s) must be determined according to other conditions and cannot be specified

programmatically.

Convert '\'s in Parameters to '/'s
Use this option to request that any backslashes (\) within the *Command Line Parameters* edit box be converted to forward slashes (/) before running the defined program.

Pass short filename to program
Use this option to request that any *Tools Macros* used in the *Command Line Parameters* edit box be converted to *short filenames* before running the defined program. This option is needed when defining a User Tool which passes a filename to a DOS program, or to a 16-bit Windows program, since these programs are typically unable to process *long filenames*.

Buttons

Save Tool button
Use the *Save Tool* button to save the current tool definition. Note that the current tool will also be saved automatically when moving to a new tool in the *Tools* list.

Clear Tool button
Use the *Clear Tool* button to clear the definition for the current tool. No confirmation is required before the tool is erased.

Test Tool Button
Use the *Test Tool* button to simulate running the current tool, without actually executing the defined program or resource. A dialog will appear showing the various fields, after the expansion of any *Tools Macros* and other requested conversions have been made.

Move up in list / Move down in list
Use these buttons to change the order in which tools appear in the User Tools submenu on the Tools menu. Clicking on a button moves the currently selected User Tool up or down in the list. Moving a User Tool up or down in the list does not cause a change in the shortcut key assignments, if any are in use. For example: a shortcut key assigned to User Tool 2 remains assigned to the second tool in the list and does not travel with a tool which is moved through that position.

Tips and Notes

The method by which Boxer runs a User Tool program makes it possible to define User Tools which are mapped to documents, rather than programs. For example, if the 'program' to be run is defined to be an HTML document, then your Internet browser will be launched to display that file. If a .DOC file is defined as the 'program' to be run, then Microsoft Word will be launched to display the document. The browse button will present a file selection dialog box which defaults to showing .EXE and .COM files, so in order to locate documents it will need to be changed to show files of all types.

The method described in the above Tip can also be used to create User Tools which map to your favorite directories. If a directory name is defined as the 'program' to

be run, then Explorer will be launched with that directory in its view. The browse button cannot be used to select a directory name, so in order to define a User Tool in this way, the directory name will need to be typed manually into the *Program* edit box.

Some DOS programs issue an on-screen report but do not pause for user interaction or confirmation before terminating. When such programs are run as User Tools, they will execute and terminate so quickly that their results cannot be studied. You can remedy this behavior by making a change to the Properties of the program being executed. Locate the program in Explorer, right-click on its icon, and select Properties. Click the *Program* tab and uncheck the option titled *Close on exit*. Click *OK* to save the change. Thereafter, when the program is run, its window will not close until its close button is clicked.

By default, Boxer is configured to present a warning message when a file it is editing is changed by another program or process. This capability is especially useful when running User Tools since it confirms that a change was made and provides the opportunity to Reload the file from disk to get the latest copy. If you will be running a User Tool which operates on the current file, this option should be kept in force. The option is located on the Configure | Preferences | Messages option page, and is titled *Warn when an edited file is changed by another program.*

If you are a user of one of the JP Software command processors and wish to specify a `.BTM` file as a User Tool, you may need to make a system configuration change before doing so. To see if a change is required, create a `.BTM` file which performs some passive operation (such as `DIR`), and try to execute it by double clicking from within Explorer. If the file executes properly, then the Windows shell is aware that `.BTM` files can be executed, and no changes are needed. If the file does not run, then a change will be needed before a `.BTM` file can be run as a program in one of Boxer's User Tools. JP Software has documented this configuration procedure in an information file on their website. At the time of writing, this file could be found at: www.jpsoft.com/help/index.htm?deskobjs.htm If the file is not found there, look in the *Support* section at www.jpsoft.com.

Here's a tip for users of JP Software's 4DOS and/or 4NT command processors: If you would like to direct the error output from a DOS program to the Windows clipboard, you can make use of the `clip:` logical device to achieve this. At the end of any *Command Line Parameters* which might be defined for a given User Tool, add the following: `>&>! clip:` This directive causes the *standard error* stream to be placed on the Windows clipboard, overwriting the current clipboard content. The clipboard can later be reviewed in Boxer or manipulated as required.

5.302 Value at Cursor

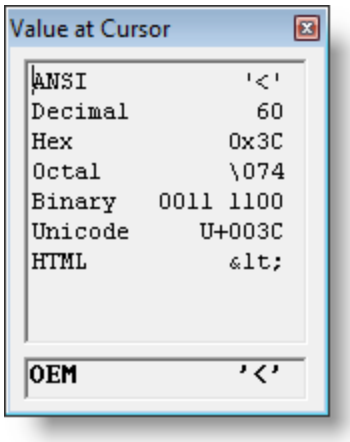
Menu: Tools > Value at Cursor

Default Shortcut Key: none

Macro function: ValueAtCursor()

The Value at Cursor command can be used to display information about the character at the text cursor. A popup window displays the character's ANSI representation and its character value in *decimal*, *hexadecimal*, *octal* and *binary* formats. The character's *Unicode* code point is also displayed. If the character has a named HTML entity, that name is displayed. Otherwise, the numeric HTML symbol is displayed.

A report for the less than symbol ('<') looks like this:



The dialog box will remain open until dismissed, and will continue to report on the character at the text cursor each time it changes.

This command can be especially useful for determining the value of characters which do not have a unique representation in the character set of the current Screen Font. For example, many ANSI fonts use an open box to represent all characters below the value 32 (Space), making it impossible to determine a character's value simply by looking at it.

The ANSI Chart and OEM Chart commands can be used to see a full listing of character values for each of these character encoding schemes.

5.303 Vertical Scroll Bar

Menu: View > Vertical Scroll Bar

Default Shortcut Key: none

Macro function: ViewVScrollBar()

The View Vertical Scroll Bar command is used to toggle on or off the scroll bar at the right edge on the editing window.

The height of the *thumb* or *scroll box* is proportional to the number of lines in the file. If the height of the thumb is one-third the height of the window, then the portion of the file visible within the window is approximately one-third of the entire file.

When the current file has insufficient lines to fill the height of the window, the Vertical Scroll Bar disappears automatically.

Clicking on the scroll bar with the right mouse button provides access to its context menu. The menu has an option to turn off display of the scroll bar.

As the *thumb* is dragged with the mouse, the current line and page count of the new view is displayed on the status bar in real-time. This makes it easier to locate a line/page of interest, since the current line need not be changed to get a report on the text that is in view.

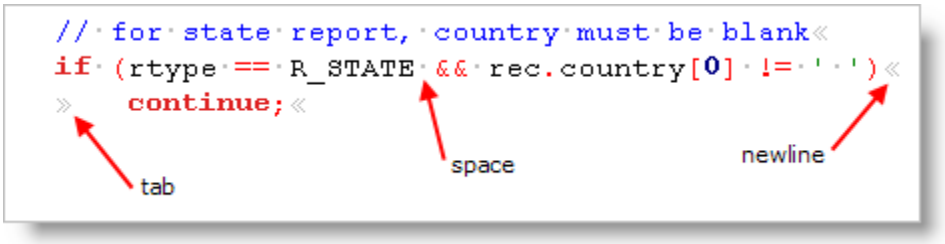
5.304 Visible Spaces

Menu: View > Visible Spaces

Default Shortcut Key: Alt+F1

Macro function: ViewVisibleSpaces()

The Visible Spaces command can be used to toggle on and off a display mode in which Spaces, Tabs and Newline characters (also known as *whitespace*) are displayed as visible symbols. This command is useful for drawing attention to extra Tabs and Spaces at the ends of lines, and to see whether indents are comprised of Tabs, Spaces, or both.



The color used to display Visible Spaces can be controlled with the Configure Colors command. In most of the default color schemes, a color has been used which makes the characters appear less prominent than foreground text. This often makes it possible to use Visible Spaces mode full-time, without concern for a cluttered display.

The symbols which are used to represent Spaces, Tabs and Newlines are user-configurable. These can be set using options on the Configure | Preferences | Display options page. Separate options are provided for use with both ANSI and OEM screen fonts.

5.305 Visual Wrap

Menu: Paragraph > Visual Wrap

Default Shortcut Key: Alt+F10

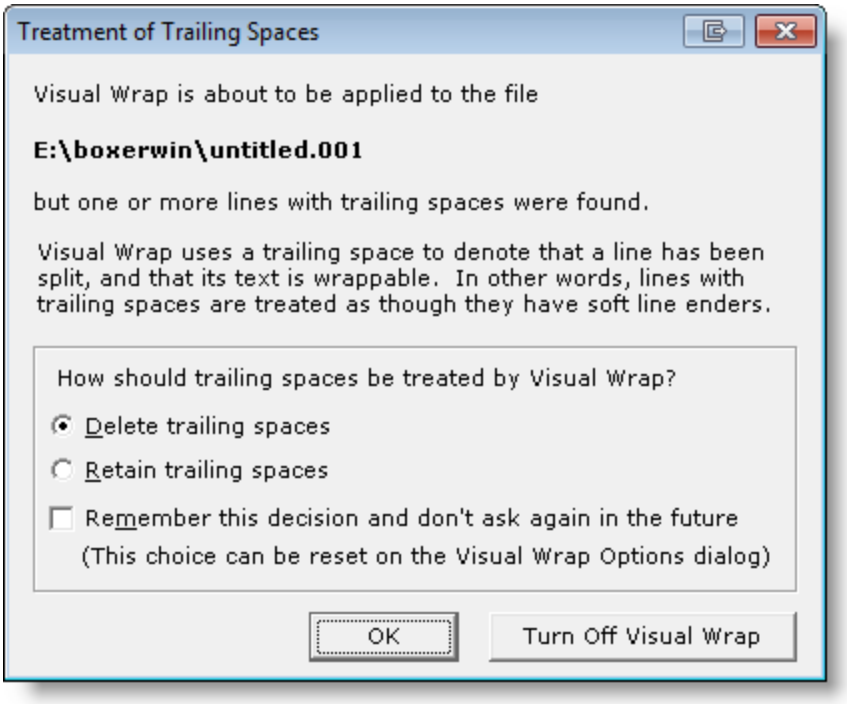
Macro function: VisualWrap()

The Visual Wrap command toggles a passive display mode that causes long lines to be wrapped to the window width (or another wrapping margin) without introducing hard line enders into the file, as would occur if the Reformat command were used. Visual Wrap is useful when editing files with very long lines that would otherwise extend off-screen to the right, out of view. It's also useful for preparing text that will later be imported into another program that prefers a long, flowing stream of text without intra-paragraph line enders.

When Visual Wrap is active, the line counter in the status bar switches to a paragraph counter, since a one-to-one relationship between screen lines and physical lines no longer exists. If Line Numbers are being displayed in the left margin, that display is also adjusted to show paragraph numbers rather than line numbers. When Visible Spaces are in use, the soft line enders on wrapped line will be denoted with a double chevron symbol (<<), while hard line enders are marked with a single chevron (<). The Go to Paragraph command can be used to jump to a paragraph by its number.

The Visual Wrap Options dialog provides access to options related to the operation of Visual Wrap. Wrapping can be set to occur at the window width, the Text Width, or at the Right Margin Rule. By default, Visual Wrap is maintained when edits are made, although this can be optionally disabled. An option is also provided for dealing with trailing spaces when Visual Wrap is first applied.

Boxer uses a trailing space to mark lines that are split or wrapped by Visual Wrap. When Visual Wrap is first applied to a file, a check is performed to see if any lines in the file already contain trailing spaces. If such lines are found, the following dialog is presented:



The nature of the file being processed will determine which option should be selected. If the trailing spaces are extraneous, the first option (delete) should be used. If the file is one in which trailing spaces are being used to mark soft line enders, the second option should be used.

☞ The Soften Line Enders command can be used to prepare a file for processing by Visual Wrap. It converts hard line enders to soft line enders, with proper consideration to paragraph boundaries, thereby making the lines of a text file flowable.

☞ The Harden Line Enders command can be used convert soft line enders to hard line enders, thereby making permanent the current on-screen formatting.

Visual Wrap Theory

When Visual Wrap is activated, any lines that are longer than the designated wrapping margin are wrapped to fit within the margin. A trailing space is left at the location where each line was split to denote that the line has a "soft" line ender. Some word processing programs insert a special character into the data stream to denote a soft line ender. As a text editor, Boxer is obliged not to introduce special characters into the text files it creates, so it simply adds a space to the end of the line. The last line of a paragraph ends with a hard line ender, and thus has no trailing space.

The use of a trailing space to mark lines with a soft line ender has several advantages:

- The space character can be seen when Visible Spaces mode is active.
- The space character will not be visible when the document is printed.
- Line enders can be easily converted between hard and soft by adding or removing the space.

Note: When Visual Wrap is applied to (or removed from) an entire file, the length of the text on each line changes, as does the total number of lines. The text itself is not changed, just its on-screen formatting. But internally, these changes cause any previously stored Undo information to become invalid. Ordinarily, Visual Wrap will be enabled just after a file is opened. If the wrapping margin is not changed thereafter, Visual Wrap will have very little impact on Undo. But if a file is frequently rewrapped or unwrapped, this side effect is something to bear in mind.

Visual Wrap mode is incompatible with Typing Wrap. If Typing Wrap is on when Visual Wrap is enabled, it will be automatically turned off.

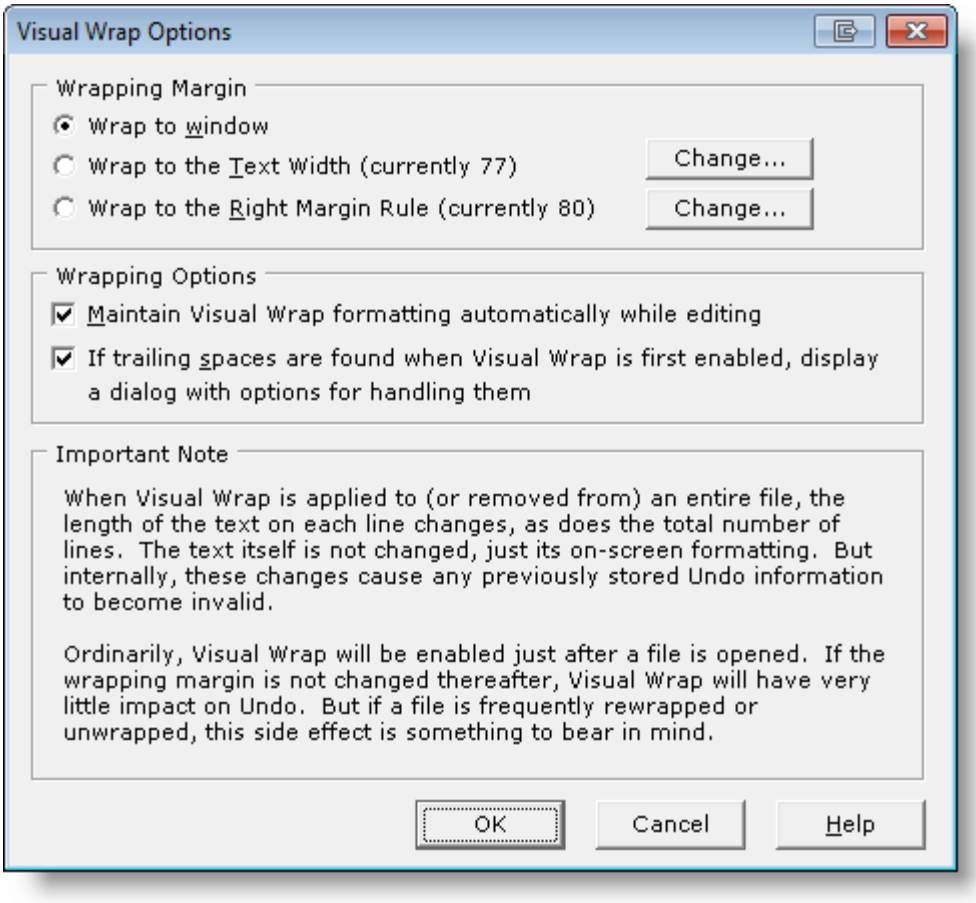
5.306 Visual Wrap Options

Menu: Paragraph > Visual Wrap Options

Default Shortcut Key: none

Macro function: VisualWrapOptions()

The Visual Wrap Options dialog contains settings that relate to the operation of Visual Wrap:



Wrapping Margin

Wrap to window
Choose this option if you want text to be wrapped to the width of the document window. The text will re-wrap automatically if the window is resized.

Wrap to the Text Width
Choose this option if you want text to be wrapped to the Text Width value that's used for various paragraph operations, such as Reformat. When this option is used, text will only rewrap when the Text Width value is changed, and not when the document window is simply resized.

Wrap to the Right Margin Rule
Choose this option if you want text to be wrapped to the Right Margin Rule value. The Right Margin Rule is an optional display feature that causes a fine vertical line to appear on-screen at the designated column. When this option is used, text will only rewrap when the Right Margin Rule value is changed, and not when the document window is simply resized.

Wrapping Options

Maintain Visual Wrap formatting automatically while editing

This is option is enabled by default. You might choose to disable this option if you find yourself distracted by the automatic wrapping that occurs when editing which causes lines to split and join automatically.

If trailing spaces are found when Visual Wrap is first enabled, display a dialog with options for handling them
When trailing spaces are found in a file to which Visual Wrap is being applied, Boxer displays a dialog with options for how these spaces should be handled. On that dialog, and option appears to disable display of the dialog. This option provides a means to re-enable the display of that dialog.

5.307 Window Close All

Menu: Window > Close All

Default Shortcut Key: none

Macro function: CloseAll()

The Close All command is used to close all open windows within the editor. If unsaved changes have been made to any file, a dialog box will appear for each such file to alert you to this fact. You will then be able to choose whether to save the changes before closing, close without saving, or cancel the Close All operation.

You can quickly tell whether a file has unsaved changes by looking for an asterisk (*) to the left of its name in the title bar or on its File Tab.

If you would prefer that Boxer be minimized automatically when the last file is closed, there is a checkbox on the Configure | Preferences | Other options page to achieve this. The option is titled *Minimize Boxer when closing last file*.

The Window | Close All command is functionally equivalent to the File | Close All command, since each file resides in its own window.

5.308 Window Last Visited

Menu: Window > Last Visited

Default Shortcut Key: Shift+Alt+F6

Macro function: WindowLastVisited()

The Window Last Visited command provides a means to return quickly to the last window that was active before the current window was activated. When a large number of files is being edited, using the Window Previous or Window Next command may not be practical for this purpose. Issuing the command repeatedly has the effect of toggling focus between the last two windows that were active.

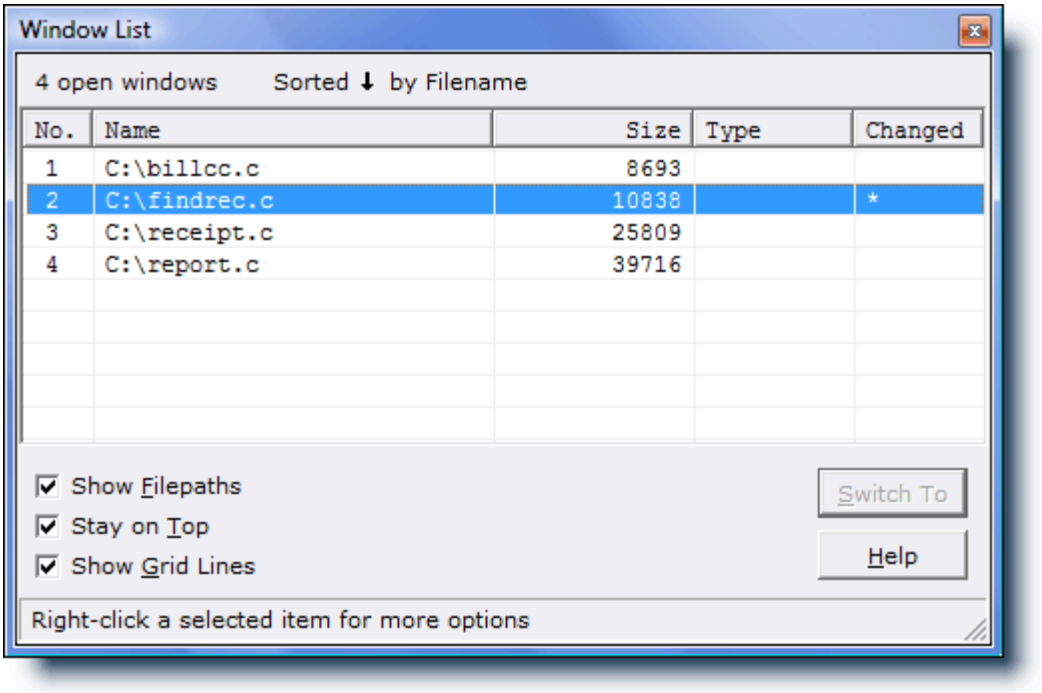
5.309 Window List

Menu: Window > Window List

Default Shortcut Key: none

Macro function: WindowList()

The Window List command presents a pop-up dialog containing the names of all currently open windows. A count of the windows is displayed at the top of the dialog.



Double-click or press *Enter* to make the selected window active. Right-clicking provides access to the Window List *context menu*. This menu provides options to Save, switch to, minimize, maximize, restore or close the selected window. The *Delete* key can be used to close the selected file.

The content of the Window List can be sorted by clicking on any of the field headers at the top of the listbox control. A second click on the same header reverses the order of the sort.

The Window List dialog is resizeable so it can accommodate long filepaths. The window is *non-modal* so that it can remain open while focus returns to an editor window.

Boxer's File Tabs also provide another method of switching among open windows, as do the Window Next and Window Previous commands.

If the Window List is left on-screen when Boxer is closed, it will be automatically reopened if the edit session is later restored.

5.310 Window Next

Menu: Window > Next

Default Shortcut Key: F6

Macro function: WindowNext()

The Window Next command is used to move to the next window when editing two or more files. When a window has been split with the Split Horizontal or Split Vertical commands, Window Next will move to the lower or right window pane, respectively. Window Next will skip over a minimized window, but the Windows-level service (*Ctrl+F6*) will stop on minimized windows.

The next window in the sequence will be determined according to the order of the File Tabs. If the File Tabs have been configured to sort the files alphabetically, Window Next will move to the window whose filename occurs next, alphabetically. If the File Tabs are not sorted, the order is determined
by Windows according to the "z order" ranking of the windows.

When many windows are open, it may prove faster to use the Window List to select a new window. Boxer's File Tabs can also be used to move quickly among windows.

5.311 Window Previous

Menu: Window > Previous

Default Shortcut Key: Shift+F6

Macro function: WindowPrevious()

The Window Previous command is used to move to the previous window when editing two or more files. When a window has been split with the Split Horizontal or Split Vertical commands, Window Next will move to the upper or left window pane, respectively. Window Previous will skip over a minimized window, but the Windows-level service (*Ctrl+F6*) will stop on minimized windows.

The previous window in the sequence will be determined according to the order of the File Tabs. If the File Tabs have been configured to sort the files alphabetically, Window Next will move to the window whose filename is previous, alphabetically. If the File Tabs are not sorted, the order is determined
by Windows according to the "z order" ranking of the windows.

When many windows are open, it may prove faster to use the Window List to select a new window. Boxer's File Tabs can also be used to move quickly among windows.

5.312 Window Skip

Menu: Window > Skip -*or*- View > File Tabs > Skip File

Default Shortcut Key: none

Macro function: WindowSkip()

The Skip command can be used to mark a file/window so that it will be skipped over by Window Previous and Window Next when these commands are used to cycle through open files. The skip status of each file is stored when an edit session is closed, so it will persist if the edit session is later resumed.

The File Tab context menu also includes options to toggle the skip state for the current file, or to set or clear the skip status for all open files.

Clicking on a file tab will cause that file's skip status to be released automatically, if the relevant option on the Configure | Preferences | Cursor dialog page is enabled.

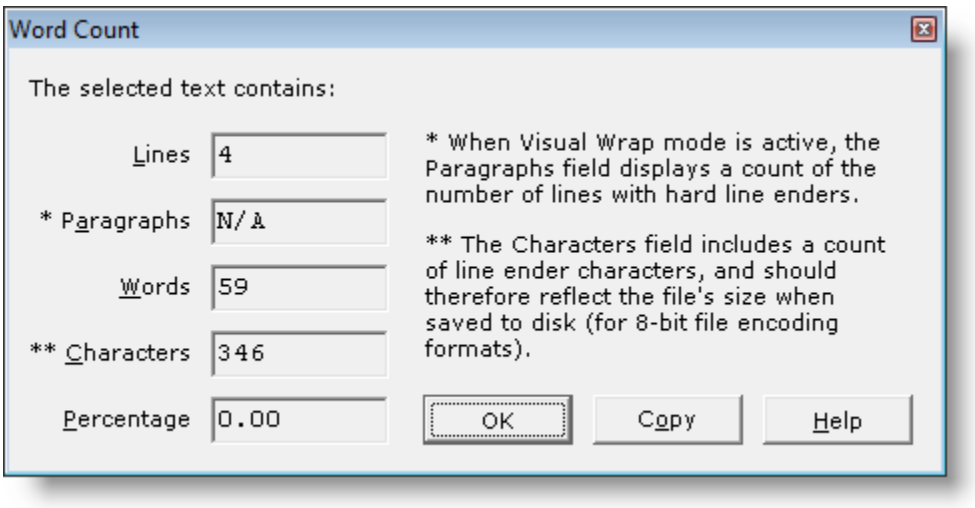
5.313 Word Count

Menu: Block > Word Count

Default Shortcut Key: None

Macro function: WordCount()

The Word Count command reports the number of lines, words and characters in the current file, or within the currently selected text. The percentage of the processed text, with respect to the whole file, is also reported.




The Word Count command reports its results using read-only edit boxes so that any of the fields can be copied to the Windows clipboard. The *Copy* button can be used to copy the fill report to the current clipboard.

6 Miscellaneous Topics

6.1 Command Line Options

Command Line Options can be used to direct Boxer to position the text cursor specially upon startup, or to activate other modes or options.

Command Line Options must be preceded with a dash (`-`) or a forward slash (`/`). Options must be placed after the filename for which they are to be applied (see example below).

The most common use for Command Line Options is to configure Boxer as the external editor for another program. For example, many development environments offer the capability to define an external editor, and provide the ability to pass the editor a starting line and/or column number. The exact syntax to achieve this integration will vary from program to program, so please consult the program's documentation.

Command Line Options

-C<*value*>
This option can be used to position the text cursor at a specified column number. The column number is specified immediately following the **C**, in decimal format. See examples below at the **-L** option flag.

-E<*value*> <filename>
Designate the end-of-record character for the next file opened. As the file is read, line breaks will be inserted after character 'value' is encountered.

-F<*value*>
This option can be used to impose a fixed length record format on the next file that appears on the command line. The record length is specified immediately following the **F**, in decimal format. As the file is read, line breaks will be inserted at column 'value'.

```
b -F512 MyFile.dat
```

Note that when the file is saved, line enders will be added. The File | Properties dialog has an option to request that line enders be removed when the file is saved.

An option to impose a fixed record length also appears on the File | Open dialog.

-G
This option can be used to disable display of Boxer's splash screen graphic upon startup.

Display of the splash screen does not slow program startup. The splash screen is removed from the screen as soon as essential startup tasks are complete. By general consensus, however, it seems the splash screen does make Boxer *appear* to take longer to start up.

-H
This option can be used to request that the next-named file on the command line be opened in read-only hex mode. If more than one file must be opened in hex mode, precede each such filename with the option flag:

```
b -H file1.txt -H file2.txt
```

-I<settings filename>
Load Boxer's settings from the named INI file. The filename is placed immediately adjacent to the **I**, without an intervening space.

-K<keyboard filename>
Load the named keyboard layout at startup. The filename is placed immediately adjacent to the **K**, without an intervening space.

-L<*value*>
This option can be used to position the text cursor at a specified line number. The line number is specified immediately following the **L**, in decimal format.

Example: to place the text cursor on line 214, column 35 the following command line could be used:

```
b myproj.cpp -L214 -C35
```

When multiple files are named, place the option flags after the filename to which they apply:

```
b myproj.cpp -L214 -C35 main.cpp -L505 -C18
```

-M<macro filename>
Run the named macro file. The macro name is placed immediately adjacent to the **M**. If the macro operates on a text file, the file to be processed should
appear to the left of the **-M** option flag so it will be opened at the time the macro is run. If two or more macros are to be applied to a single file, use **-M** repeatedly for each macro:

```
b file.txt -Mmacro1.bm -Mmacro2.bm
```

<filename> -O<value>
Position the text cursor at a specified byte offset in the immediately preceding filename. The byte offset is specified immediately following the **O**, in hexadecimal (use a leading 'x') or decimal format.

-P<project filename>
Open the named project file. The project name is placed immediately adjacent to the **P**.

If a filepath is not supplied, the file will be opened from the Projects directory. For complete information about project files, see the Project | New command.

-Q
This option can be used to force Boxer into quiet mode. All use of sound will be disabled for the current editing session, and for all future sessions. Sounds can be enabled on the Configure | Preferences | Messages options page.


This option would only be needed if Boxer should fail to start due to a non-functioning sound card. In such case the normal method for disabling sounds, via the Configure | Preferences | Messages dialog, would be unavailable.

-R <filename>
Open the next file that appears on the command line in read-only mode.


This option might be employed when Boxer is to be used as an external file viewer.

-S<*text*>
This option can be used to position the text cursor at the first occurrence of the specified search string. The search string is entered immediately following the **S**. Regular Expressions are not permitted in this context; the search performed is a simple text search.

Example: to place the text cursor at the first occurrence of the word 'porcelain' the following command line could be used:

```
b oldsigns.txt -Sporcelain
```

-T<tab settings> <filename>
Specify tab settings for the next file, and all other files named on the command line. If a single value 'n' is present, fixed width tabs of size 'n' will be used. If a comma-delimited list of values is present, variable width tab stops will be assigned at the designated columns. Command line tab settings take precedence over default tab settings, and over tab settings specified in `Syntax.ini`.

Example 1: open the named files with a fixed width tab size of 8:

```
b -T8 file1.txt file2.txt
```

Example 2: open the named file with tab stops at the columns indicated:

```
b -T4,12,22,30,44,60 orders.dat
```

-**WH<value>**
Set the height of the main application window to the value specified.

```
b -WH580 file1.txt file2.txt
```

-**WL<value>**

Position the main application window with its left edge at the specified pixel location.

```
b -WL50 file1.txt file2.txt
```

-**WT<value>**
Position the main application window with its top edge at the specified pixel location.

```
b -WT75 file1.txt file2.txt
```

-**WW<value>**
Set the width of the main application window to the value specified.

```
b -WW820 file1.txt file2.txt
```

-**W0**
Turn Typing Wrap off for the immediately preceding filename. The following command line will turn off Typing Wrap for file1.txt, but not for file2.txt.

```
b file1.txt -W0 file2.txt
```

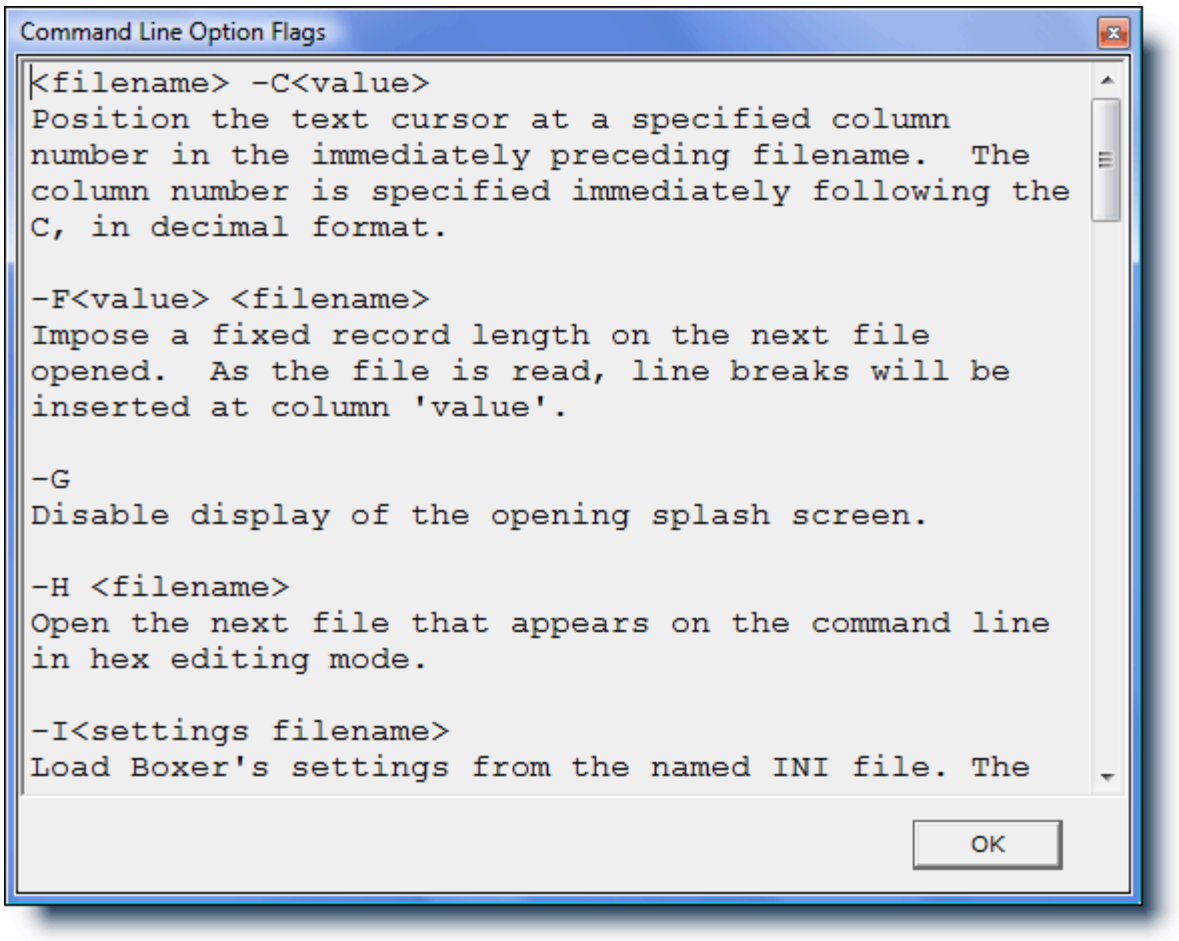
-W1
Turn Typing Wrap on for the immediately preceding filename. The following command line will turn on Typing Wrap for `file1.txt`, but not for `file2.txt`.

```
b file1.txt -W1 file2.txt
```

-1
Force filenames to be added to an existing session, regardless of the current setting for Multiple Instances. The option can appear anywhere on the command line, and applies to all files on the command line.

-2
Force filenames to be opened in a new session, regardless of the current setting for Multiple Instances. The option can appear anywhere on the command line, and applies to all files on the command line.

-?
This option causes a pop-up window to appear that displays all of Boxer's command line options:



Wildcard Expressions

Boxer supports the use of wildcard expressions to match an entire class of files. All files matched by the wildcard expression will be opened for editing. The use of the asterisk (`*`) and question mark (`?`) are supported. For example, to edit all files in the current directory with a `.cpp` file extension, the following command line could be used:

```
b *.cpp
```

To edit all files named `report` with a single character file extension, use this wildcard expression:

```
b report.?
```

6.2 Converting CSV Data to Fixed Width Format

Boxer is a good tool for working with data files that use comma-separated value (CSV) or fixed width format. Its ability to select and manipulate text in rectangular columns is especially useful in this regard. Also, the Shaded Tab Zones feature allows fixed width data files to be viewed in a mode that aids in visually distinguishing between data

columns.

Sometimes it may be desirable to convert CSV formatted data to fixed width format. Boxer does not have a built-in command to make this conversion directly, but such a conversion can be accomplished by using the following steps:

1. Load the CSV file for editing
2. Issue the View | Tab Display Size command
3. Check the checkbox entitled "*In addition to tab*..."
4. Make sure a comma character appears in the *Additional tab character* box
5. Click the *Intelli-Tabs* button
6. Click *OK*

The comma has now been designated as an additional tab character, so that the CSV file is displayed on-screen with its fields properly aligned. A series of variable tab stops, auto-computed by the Intelli-Tabs feature, are being used to align the fields properly. All that remains is to convert the pseudo tabs (commas) to spaces in the data itself in accordance with the display settings:

7. Issue the Edit | Select All Text command
8. Issue the Block | Convert Other | Tabs to Spaces command

The CSV file is now in fixed width format, with one extra space between each field. If you prefer extra spaces, or no space at all, you can reload the original file and perform the operation again, manually adjusting the variable tab stop values on the Tab Display Size dialog before proceeding with the conversion.

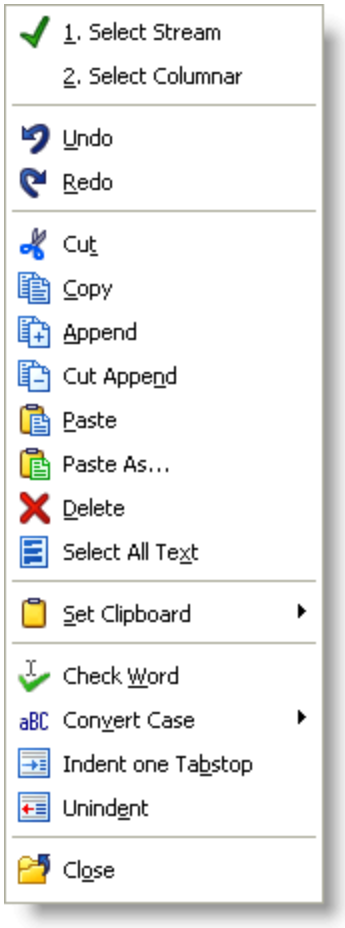

The steps above presume that the data does not contain embedded commas or tabs characters. If these characters are present, special attention will be required before this method can be used.

6.3 Context Menu

Menu: not applicable

Default Shortcut Key: Shift+F10

Boxer provides a *context menu* which can be accessed by clicking the right mouse button anywhere within the active editor window. The Context Menu contains a group of commands which are likely to be useful while editing text. Using the Context Menu is typically faster than using the mouse to select a command from the main menu, since the context menu will pop-up at the point where the mouse is clicked.



The operation of the commands which appear on the Context Menu is identical to that of their main menu counterparts. The Context Menu simply provides an alternative method of issuing those commands.

In Boxer's default keyboard layout, the Context Menu is assigned to the *Shift+F10* key sequence. The *Open Context Menu* command is not one which appears in the main menu, but its assignment can be changed with Configure | Keyboard command.

6.4 Cursor Movement Commands

Cursor Movement Commands

Boxer has a variety of commands for moving the text cursor throughout the edited file. Most of these commands do not appear within the main menu, but are summarized in the table below along with their default key assignments:

Command	Default Key Assignment	Alternative
Up	Up Arrow	
Down	Down Arrow	

Left	Left Arrow	
Right	Right Arrow	
Word Left	Ctrl+Left Arrow	
Word Right	Ctrl+Right Arrow	
Next Paragraph	none ②	
Previous Paragraph	none ②	
Page Up	PgUp	
Page Down	PgDn	
Start of Line	Home	
End of Line	End	
Top of Page	Ctrl+PgUp	Home - Home ①
Bottom of Page	Ctrl+PgDn	End - End ①
Start of File	Ctrl+Home	Home - Home - Home ①
End of File	Ctrl+End	End - End - End ①
Left Window Edge	none ②	
Right Window Edge	none ②	
Page Left	none ②	
Page Right	none ②	
Scroll Up	none ②	
Scroll Down	none ②	
Scroll Left	Alt+Left	
Scroll Right	Alt+Right	
Backtab	Ctrl+Alt+Left	

① These assignments are available when *Home-Home-Home* and *End-End-End* operation is enabled. Options are available on the Configure | Preferences | Cursor Travel options page.

② Though these commands do not have default key assignments, the Configure | Keyboard dialog can be used to give them assignments if desired.

Text Selection Commands

Most cursor movement commands will respond to the *Shift* key as an 'accelerator' to either initiate a text selection, or to extend a text selection that already exists. The table below summarizes these commands:

Command	Default Key Assignment	Alternative
Select Up	Shift+Up Arrow	
Select Down	Shift+Down Arrow	
Select Left	Shift+Left Arrow	
Select Right	Shift+Right Arrow	
Select Word Left	Shift+Ctrl+Left Arrow	
Select Word Right	Shift+Ctrl+Right Arrow	
Select Page Up	Shift+PgUp	
Select Page Down	Shift+PgDn	
Select to Start of Line	Shift+Home	
Select to End of Line	Shift+End	
Select to Top of Page	Shift+Ctrl+PgUp	Shift+Home - Shift+Home
Select Bottom of Page	Shift+Ctrl+PgDn	Shift+End - Shift+End
Select to Start of File	Shift+Ctrl+Home	Shift+Home - Shift+Home - Shift+Home
Select to End of File	Shift+Ctrl+End	Shift+End - Shift+End - Shift+End
Select Page Left	no assignment	
Select Page Right	no assignment	

6.5 Default Key Assignments (command order)

This list was generated using the *Make List* button in the Configure | Keyboard dialog box. The same procedure can be used to create a list for other supplied keyboard layouts, or for layouts you create yourself.

About Boxer	No Assignment
Active Spell Checking	Alt+F7
Align Center	Ctrl+F8
Align Left	Ctrl+F7
Align Right	Ctrl+F9
Align Smooth	Ctrl+F11
ANSI Chart	No Assignment
ANSI to OEM	No Assignment

Append	Shift+Ctrl+C
Apply Highlighting	No Assignment
Arrange Icons	No Assignment
ASCII to EBCDIC	No Assignment
Auto-Complete	Ctrl+Space
Auto-Complete List	Alt+Down
Auto-Number	No Assignment
Backspace	BkSp
Backtab	Ctrl+Alt+Left
Bookmark Manager	Shift+F9
Bottom of Page	Ctrl+PgDn
Boxer Shorts	No Assignment
Boxer Software Website	No Assignment
Bring User Lists to Top	No Assignment
Calculator	F11
Calendar	No Assignment
Cancel	Esc
Cascade	No Assignment
Cascade Horizontal	No Assignment
Cascade Vertical	No Assignment
Check for Latest Version	No Assignment
Check Word	Shift+F7
Clear All Bookmarks	No Assignment
Clear All Clipboards	No Assignment
Clear Clipboard 1	No Assignment
Clear Clipboard 2	No Assignment
Clear Clipboard 3	No Assignment
Clear Clipboard 4	No Assignment
Clear Clipboard 5	No Assignment
Clear Clipboard 6	No Assignment
Clear Clipboard 7	No Assignment
Clear Clipboard 8	No Assignment
Clear Recent Files List	No Assignment

Clear Recent Projects List	No Assignment
Clear Undo	No Assignment
Clear Windows Clipboard	No Assignment
Close	Alt+X
Close All	Ctrl+Alt+X
Close All but Active	No Assignment
Color Chart	No Assignment
Command Multiplier	Alt+Y
Comment	No Assignment
Configure Auto-Complete	No Assignment
Configure Colors	No Assignment
Configure Ctags Function Index	No Assignment
Configure Keyboard	No Assignment
Configure Preferences	No Assignment
Configure Printer Font	No Assignment
Configure Screen Font	No Assignment
Configure Syntax Highlighting	No Assignment
Configure Templates	No Assignment
Configure Text Highlighting	No Assignment
Configure Toolbar	No Assignment
Configure User Tools	No Assignment
Contact Information	No Assignment
Convert Case Invert	No Assignment
Convert Case Lower	No Assignment
Convert Case Sentences	No Assignment
Convert Case Title	No Assignment
Convert Case Upper	No Assignment
Convert Case Words	No Assignment
Copy	Ctrl+C
Copy	Ctrl+Ins
Copy Filename	No Assignment
Ctags Function Index	No Assignment
Cut	Ctrl+X

Cut	Shift+Del
Cut Append	Shift+Ctrl+X
Declaration	No Assignment
Decrement	No Assignment
Delete	Del
Delete Blank Lines	No Assignment
Delete Bookmarked Lines	No Assignment
Delete Current Line	Alt+D
Delete Duplicate Lines	No Assignment
Delete Lines that Begin with	No Assignment
Delete Lines that Contain	No Assignment
Delete Lines that do not Begin with No Assignment	
Delete Lines that do not Contain	No Assignment
Delete Lines that do not End with No Assignment	
Delete Lines that End with	No Assignment
Delete Next Word	Ctrl+Del
Delete Previous Word	Ctrl+BkSp
Delete to End of Line	No Assignment
Delete to Start of Line	Ctrl+K
Divide	No Assignment
Down	Down
Duplicate & Increment	Shift+F2
Duplicate Line	F2
Duplicate Line	Alt+O
EBCDIC to ASCII	No Assignment
Edit Clipboard 1	No Assignment
Edit Clipboard 2	No Assignment
Edit Clipboard 3	No Assignment
Edit Clipboard 4	No Assignment
Edit Clipboard 5	No Assignment
Edit Clipboard 6	No Assignment
Edit Clipboard 7	No Assignment

Edit Clipboard 8	No Assignment
Edit Windows Clipboard	No Assignment
Email Boxer Software	No Assignment
End of File	Ctrl+End
End of Line	End
Error Chart	No Assignment
Exit	Alt+F4
Exit	Alt+Q
Explore Data Folder	No Assignment
Explore Program Folder	No Assignment
FAQs	No Assignment
Fast Frame	Alt+F12
File Picker	Alt+K
File Properties	No Assignment
File Tabs Bottom	No Assignment
File Tabs Top	No Assignment
Fill with String	No Assignment
Find	Ctrl+F
Find a Disk File	No Assignment
Find and Count	No Assignment
Find Differing Lines	Ctrl+D
Find Distinct Lines	No Assignment
Find Duplicate Lines	No Assignment
Find Fast	Ctrl+F3
Find Mate	Ctrl+]
Find Next	F3
Find Previous	Shift+F3
Find Text in Disk Files	No Assignment
Find Unique Lines	No Assignment
Flip Case	Shift+Ctrl+F
FTP Open	Shift+Alt+O
FTP Save As	Shift+Alt+F12
Go to Byte Offset	No Assignment

Go to Column	Shift+Ctrl+G
Go to Line	Ctrl+G
Go to Line	Alt+G
Help	F1
Help On	Shift+F1
Hex Mode	Shift+Alt+X
HTML Image Tag	No Assignment
Increment	No Assignment
Indent one Space	No Assignment
Indent one Tabstop	Shift+Tab
Indent with String	No Assignment
Insert Character	No Assignment
Insert File	Ctrl+I
Insert File	Alt+I
Insert Filename	No Assignment
Insert Formfeed	No Assignment
Insert Line Above	Shift+Ctrl+Enter
Insert Line Below	Ctrl+Enter
Insert Long Date	Shift+Ctrl+F11
Insert Long Time	Shift+Ctrl+F12
Insert Short Date	Shift+F11
Insert Short Time	Shift+F12
Insert Symbol 1	Shift+Ctrl+1
Insert Symbol 2	Shift+Ctrl+2
Insert Symbol 3	Shift+Ctrl+3
Insert Symbol 4	Shift+Ctrl+4
Insert Symbol 5	Shift+Ctrl+5
Insert Symbol 6	Shift+Ctrl+6
Insert Symbol 7	Shift+Ctrl+7
Insert Symbol 8	Shift+Ctrl+8
Insert Tab	Tab
Invert Lines	No Assignment
Justification Style	Ctrl+J

Left	Left
Left Window Edge	No Assignment
Line Drawing	Ctrl+F12
Load Key Recording	No Assignment
Macros	F8
Make Line Bottom	No Assignment
Make Line Center	Center 5
Make Line Top	No Assignment
Maximize	No Assignment
Maximize All	No Assignment
Minimize	No Assignment
Minimize All	No Assignment
Multiply	No Assignment
New	Ctrl+N
Next Bookmark	Shift+Ctrl+Down
Next Function	Ctrl+Alt+Down
Next Paragraph	No Assignment
OEM Chart	No Assignment
OEM to ANSI	No Assignment
Open	Ctrl+O
Open Context Menu	Shift+F10
Open Email at Cursor	Ctrl+E
Open File in Browser	Ctrl+B
Open Filename at Cursor	Ctrl+L
Open Header File	Ctrl+H
Open Hex Mode	Ctrl+Alt+O
Open Program at Cursor	No Assignment
Open Recent File 1	No Assignment
Open Recent File 2	No Assignment
Open Recent File 3	No Assignment
Open Recent File 4	No Assignment
Open Recent File 5	No Assignment
Open Recent File 6	No Assignment

Open Recent File 7	No Assignment
Open Recent File 8	No Assignment
Open Recent File 9	No Assignment
Open Recent File 10	No Assignment
Open Recent File 11	No Assignment
Open Recent File 12	No Assignment
Open Recent File 13	No Assignment
Open Recent File 14	No Assignment
Open Recent File 15	No Assignment
Open Recent File 16	No Assignment
Open Recent File 17	No Assignment
Open Recent File 18	No Assignment
Open Recent File 19	No Assignment
Open Recent File 20	No Assignment
Open Recent File 21	No Assignment
Open Recent File 22	No Assignment
Open Recent File 23	No Assignment
Open Recent File 24	No Assignment
Open Recent Project 1	No Assignment
Open Recent Project 2	No Assignment
Open Recent Project 3	No Assignment
Open Recent Project 4	No Assignment
Open Recent Project 5	No Assignment
Open Recent Project 6	No Assignment
Open Recent Project 7	No Assignment
Open Recent Project 8	No Assignment
Open Recent Project 9	No Assignment
Open Recent Project 10	No Assignment
Open Recent Project 11	No Assignment
Open Recent Project 12	No Assignment
Open Recent Project 13	No Assignment
Open Recent Project 14	No Assignment
Open Recent Project 15	No Assignment

Open Recent Project 16	No Assignment
Open System Files	No Assignment
Open URL at Cursor	Ctrl+U
Order Boxer	No Assignment
Order with PayPal	No Assignment
Page Down	PgDn
Page Left	No Assignment
Page Right	No Assignment
Page Setup	No Assignment
Page Up	PgUp
Paste	Ctrl+V
Paste	Shift+Ins
Paste As	Shift+Ctrl+V
Paste Clipboard 1	Shift+Alt+1
Paste Clipboard 2	Shift+Alt+2
Paste Clipboard 3	Shift+Alt+3
Paste Clipboard 4	Shift+Alt+4
Paste Clipboard 5	Shift+Alt+5
Paste Clipboard 6	Shift+Alt+6
Paste Clipboard 7	Shift+Alt+7
Paste Clipboard 8	Shift+Alt+8
Paste Windows Clipboard	Shift+Alt+0
Pause Recording	No Assignment
Playback Keys	F5
Previous Bookmark	Shift+Ctrl+Up
Previous Function	Ctrl+Alt+Up
Previous Paragraph	No Assignment
Print All Color	No Assignment
Print All Monochrome	No Assignment
Print All Normal	No Assignment
Print Color	No Assignment
Print Monochrome	No Assignment
Print Normal	Ctrl+P

Print Preview Color	No Assignment
Print Preview Monochrome	No Assignment
Print Preview Normal	No Assignment
Print Setup	No Assignment
Project Add All	No Assignment
Project Add One	No Assignment
Project Auto-Update	No Assignment
Project Close	No Assignment
Project Delete	No Assignment
Project Edit Active	No Assignment
Project Edit Other	No Assignment
Project New	No Assignment
Project Open	No Assignment
Project Remove	No Assignment
Project Update All	No Assignment
Project Update One	No Assignment
Quote and Reformat	Ctrl+Q
Record Keys	Shift+F5
Redo	Ctrl+Y
Redo All	No Assignment
Reference	No Assignment
Reformat	Ctrl+F10
Reload File	Shift+Ctrl+O
Repeat Last Command	F10
Replace	Ctrl+R
Replace	Alt+R
Replace Again	Shift+Ctrl+R
Replace Line Enders	Ctrl+Alt+R
Restore	No Assignment
Restore All	No Assignment
Right	Right
Right Window Edge	No Assignment
Run Macro 1	No Assignment

Run Macro 2	No Assignment
Run Macro 3	No Assignment
Run Macro 4	No Assignment
Run Macro 5	No Assignment
Run Macro 6	No Assignment
Run Macro 7	No Assignment
Run Macro 8	No Assignment
Run Macro 9	No Assignment
Run Macro 10	No Assignment
Run Macro 11	No Assignment
Run Macro 12	No Assignment
Run Macro 13	No Assignment
Run Macro 14	No Assignment
Run Macro 15	No Assignment
Run Macro 16	No Assignment
Run Macro 17	No Assignment
Run Macro 18	No Assignment
Run Macro 19	No Assignment
Run Macro 20	No Assignment
Run Macro 21	No Assignment
Run Macro 22	No Assignment
Run Macro 23	No Assignment
Run Macro 24	No Assignment
Run Macro 25	No Assignment
Run Macro 26	No Assignment
Run Macro 27	No Assignment
Run Macro 28	No Assignment
Run Macro 29	No Assignment
Run Macro 30	No Assignment
Run Macro 31	No Assignment
Run Macro 32	No Assignment
Run Macro 33	No Assignment
Run Macro 34	No Assignment

Run Macro 35	No Assignment
Run Macro 36	No Assignment
Run Macro 37	No Assignment
Run Macro 38	No Assignment
Run Macro 39	No Assignment
Run Macro 40	No Assignment
Run Macro 41	No Assignment
Run Macro 42	No Assignment
Run Macro 43	No Assignment
Run Macro 44	No Assignment
Run Macro 45	No Assignment
Run Macro 46	No Assignment
Run Macro 47	No Assignment
Run Macro 48	No Assignment
Run Macro 49	No Assignment
Run Macro 50	No Assignment
Save	Ctrl+S
Save a Copy As	No Assignment
Save All	Shift+Ctrl+S
Save As	F12
Save Key Recording	No Assignment
Save Selection As	No Assignment
Scroll Down	Ctrl+Up
Scroll Left	Alt+Left
Scroll Right	Alt+Right
Scroll Up	Ctrl+Down
Select All Text	Ctrl+A
Select Columnar	Alt+2
Select Down	Shift+Down
Select Left	Shift+Left
Select Page Down	Shift+PgDn
Select Page Left	No Assignment
Select Page Right	No Assignment

Select Page Up	Shift+PgUp
Select Right	Shift+Right
Select Stream	Alt+1
Select to Bottom of Page	Shift+Ctrl+PgDn
Select to End of File	Shift+Ctrl+End
Select to End of Line	Shift+End
Select to Start of File	Shift+Ctrl+Home
Select to Start of Line	Shift+Home
Select to Top of Page	Shift+Ctrl+PgUp
Select Up	Shift+Up
Select without Shift	Alt+M
Select Word Left	Shift+Ctrl+Left
Select Word Right	Shift+Ctrl+Right
Set Clipboard 1	Ctrl+1
Set Clipboard 2	Ctrl+2
Set Clipboard 3	Ctrl+3
Set Clipboard 4	Ctrl+4
Set Clipboard 5	Ctrl+5
Set Clipboard 6	Ctrl+6
Set Clipboard 7	Ctrl+7
Set Clipboard 8	Ctrl+8
Set Clipboard Next	No Assignment
Set Clipboard Previous	No Assignment
Set Windows Clipboard	Ctrl+0
Shaded Tab Zones	No Assignment
Sort File Tabs by Extension	No Assignment
Sort File Tabs by Name	No Assignment
Sort File Tabs by Use	No Assignment
Sort Lines	No Assignment
Spaces to Tabs	No Assignment
Spell Checker	F7
Split Horizontal	No Assignment
Split Vertical	No Assignment

Start of File	Ctrl+Home
Start of Line	Home
Strip HTML/XML Tags	No Assignment
Strip Leading Spaces	No Assignment
Strip Trailing Spaces	No Assignment
Swap Lines	F4
Swap Words	Shift+F4
Switch to Window 1	No Assignment
Switch to Window 2	No Assignment
Switch to Window 3	No Assignment
Switch to Window 4	No Assignment
Switch to Window 5	No Assignment
Switch to Window 6	No Assignment
Switch to Window 7	No Assignment
Switch to Window 8	No Assignment
Switch to Window 9	No Assignment
Synchronized Scroll	No Assignment
Syntax Highlight As	No Assignment
Tab Display Size	Alt+F9
Tabs to Spaces	No Assignment
Technical Support	No Assignment
Templates	Ctrl+T
Text Width	Ctrl+W
Tile Across	No Assignment
Tile Down	No Assignment
Toggle Bookmark	F9
Toggle Edit Mode	Ins
Toggle Read-Only	No Assignment
Toolbar Bottom	No Assignment
Toolbar Left	No Assignment
Toolbar Right	No Assignment
Toolbar Top	No Assignment
Top of Page	Ctrl+PgUp

Typing Wrap	Ctrl+F5
Total and Average	No Assignment
Uncomment	No Assignment
Undo	Ctrl+Z
Undo	Alt+U
Undo All	No Assignment
Unformat	Ctrl+Alt+F10
Unindent	Shift+BkSp
Up	Up
User List 1 (Two-Letter U.S. State Codes) No Assignment	
User List 2 (Internet Country Codes) No Assignment	
User List 3 (Hi-Tech Stock Symbols) No Assignment	
User List 4 (HTML Tags)	No Assignment
User List 5 (Fahrenheit / Celsius Table) No Assignment	
User List 6 (Metric/English Conversions) No Assignment	
User List 7 (Email and URL Addresses) No Assignment	
User List 8 (undefined)	No Assignment
User Tool 1 (Example: Google)	No Assignment
User Tool 2 (Example: Wikipedia) No Assignment	
User Tool 3 (Example: Dictionary) No Assignment	
User Tool 4 (Example: Weather)	No Assignment
User Tool 5	No Assignment
User Tool 6	No Assignment
User Tool 7 (undefined)	No Assignment
User Tool 8 (undefined)	No Assignment
User Tool 9 (undefined)	No Assignment
User Tool 10 (undefined)	No Assignment
User Tool 11 (undefined)	No Assignment

User Tool 12 (undefined)	No Assignment
User Tool 13 (undefined)	No Assignment
User Tool 14 (undefined)	No Assignment
User Tool 15 (undefined)	No Assignment
User Tool 16 (undefined)	No Assignment
User Tool 17 (undefined)	No Assignment
User Tool 18 (undefined)	No Assignment
User Tool 19 (undefined)	No Assignment
User Tool 20 (undefined)	No Assignment
User Tool 21 (undefined)	No Assignment
User Tool 22 (undefined)	No Assignment
User Tool 23 (undefined)	No Assignment
User Tool 24 (undefined)	No Assignment
Value at Cursor	No Assignment
View Bookmarks	Alt+F2
View File Tabs	No Assignment
View Hex Ruler	No Assignment
View Horizontal Scroll Bar	No Assignment
View Line Numbers	Alt+F3
View Right Margin Rule	Alt+F6
View Status Bar	No Assignment
View Syntax Highlighting	No Assignment
View Text Highlighting	Alt+F8
View Text Ruler	Alt+F5
View Toolbar	No Assignment
View Vertical Scroll Bar	No Assignment
View Visible Spaces	Alt+F1
Window List	No Assignment
Window Next	F6
Window Previous	Shift+F6
Window Skip	No Assignment
Word Count	No Assignment
Word Left	Ctrl+Left

Word Right	Ctrl+Right

6.6 Default Key Assignments (key order)

This list was generated using the *Make List* button in the Configure | Keyboard dialog box, and then sorted by key names. The same procedure can be used to create a list for other supplied keyboard layouts, or for layouts you create yourself.

Select Stream	Alt+1
Select Columnar	Alt+2
Delete Current Line	Alt+D
Auto-Complete List	Alt+Down
View Visible Spaces	Alt+F1
Fast Frame	Alt+F12
View Bookmarks	Alt+F2
View Line Numbers	Alt+F3
Exit	Alt+F4
View Text Ruler	Alt+F5
View Right Margin Rule	Alt+F6
Active Spell Checking	Alt+F7
View Text Highlighting	Alt+F8
Tab Display Size	Alt+F9
Go to Line	Alt+G
Insert File	Alt+I
File Picker	Alt+K
Scroll Left	Alt+Left
Select without Shift	Alt+M
Duplicate Line	Alt+O
Exit	Alt+Q
Replace	Alt+R
Scroll Right	Alt+Right
Undo	Alt+U
Close	Alt+X
Command Multiplier	Alt+Y

Backspace	BkSp
Make Line Center	Center 5
Set Windows Clipboard	Ctrl+0
Set Clipboard 1	Ctrl+1
Set Clipboard 2	Ctrl+2
Set Clipboard 3	Ctrl+3
Set Clipboard 4	Ctrl+4
Set Clipboard 5	Ctrl+5
Set Clipboard 6	Ctrl+6
Set Clipboard 7	Ctrl+7
Set Clipboard 8	Ctrl+8
Select All Text	Ctrl+A
Next Function	Ctrl+Alt+Down
Unformat	Ctrl+Alt+F10
Backtab	Ctrl+Alt+Left
Open Hex Mode	Ctrl+Alt+O
Replace Line Enders	Ctrl+Alt+R
Previous Function	Ctrl+Alt+Up
Close All	Ctrl+Alt+X
Open File in Browser	Ctrl+B
Delete Previous Word	Ctrl+BkSp
Copy	Ctrl+C
Find Differing Lines	Ctrl+D
Delete Next Word	Ctrl+Del
Scroll Up	Ctrl+Down
Open Email at Cursor	Ctrl+E
End of File	Ctrl+End
Insert Line Below	Ctrl+Enter
Find	Ctrl+F
Reformat	Ctrl+F10
Align Smooth	Ctrl+F11
Line Drawing	Ctrl+F12
Find Fast	Ctrl+F3

Typing Wrap	Ctrl+F5
Align Left	Ctrl+F7
Align Center	Ctrl+F8
Align Right	Ctrl+F9
Go to Line	Ctrl+G
Open Header File	Ctrl+H
Start of File	Ctrl+Home
Insert File	Ctrl+I
Copy	Ctrl+Ins
Justification Style	Ctrl+J
Delete to Start of Line	Ctrl+K
Open Filename at Cursor	Ctrl+L
Word Left	Ctrl+Left
New	Ctrl+N
Open	Ctrl+O
Print Normal	Ctrl+P
Bottom of Page	Ctrl+PgDn
Top of Page	Ctrl+PgUp
Quote and Reformat	Ctrl+Q
Replace	Ctrl+R
Word Right	Ctrl+Right
Save	Ctrl+S
Auto-Complete	Ctrl+Space
Templates	Ctrl+T
Open URL at Cursor	Ctrl+U
Scroll Down	Ctrl+Up
Paste	Ctrl+V
Text Width	Ctrl+W
Cut	Ctrl+X
Redo	Ctrl+Y
Undo	Ctrl+Z
Find Mate	Ctrl+]
Delete	Del

Down	Down
End of Line	End
Cancel	Esc
Help	F1
Repeat Last Command	F10
Calculator	F11
Save As	F12
Duplicate Line	F2
Find Next	F3
Swap Lines	F4
Playback Keys	F5
Window Next	F6
Spell Checker	F7
Macros	F8
Toggle Bookmark	F9
Start of Line	Home
Toggle Edit Mode	Ins
Left	Left
Cascade Horizontal	No Assignment
Configure Syntax Highlighting	No Assignment
Delete Lines that End with	No Assignment
Delete to End of Line	No Assignment
Apply Highlighting	No Assignment
Clear Clipboard 8	No Assignment
Clear Recent Projects List	No Assignment
Edit Clipboard 5	No Assignment
Configure Templates	No Assignment
Project Close	No Assignment
Clear Clipboard 6	No Assignment
Calendar	No Assignment
Delete Lines that Contain	No Assignment
Delete Lines that do not Begin withNo Assignment	
Edit Clipboard 4	No Assignment

Cascade	No Assignment
File Properties	No Assignment
Edit Clipboard 7	No Assignment
Configure Text Highlighting	No Assignment
Configure Toolbar	No Assignment
ANSI Chart	No Assignment
Edit Clipboard 2	No Assignment
Edit Clipboard 3	No Assignment
Find Text in Disk Files	No Assignment
Arrange Icons	No Assignment
Clear Clipboard 5	No Assignment
Run Macro 21	No Assignment
Edit Clipboard 8	No Assignment
Declaration	No Assignment
Clear Clipboard 2	No Assignment
Run Macro 42	No Assignment
Clear Clipboard 3	No Assignment
File Tabs Bottom	No Assignment
Find Duplicate Lines	No Assignment
Edit Clipboard 1	No Assignment
Delete Lines that Begin with	No Assignment
Increment	No Assignment
Indent one Space	No Assignment
Delete Lines that do not Contain	No Assignment
Delete Lines that do not End with No Assignment	
Insert Character	No Assignment
Cascade Vertical	No Assignment
File Tabs Top	No Assignment
Insert Filename	No Assignment
Divide	No Assignment
Clear All Clipboards	No Assignment
Convert Case Lower	No Assignment
Find Distinct Lines	No Assignment

Boxer Shorts	No Assignment
EBCDIC to ASCII	No Assignment
Project Remove	No Assignment
Switch to Window 1	No Assignment
Switch to Window 2	No Assignment
Switch to Window 3	No Assignment
Switch to Window 4	No Assignment
Switch to Window 5	No Assignment
Clear Undo	No Assignment
Switch to Window 7	No Assignment
Switch to Window 8	No Assignment
Switch to Window 9	No Assignment
Decrement	No Assignment
Clear Clipboard 1	No Assignment
Strip Leading Spaces	No Assignment
Strip Trailing Spaces	No Assignment
Clear Clipboard 4	No Assignment
Delete Duplicate Lines	No Assignment
HTML Image Tag	No Assignment
Configure Keyboard	No Assignment
Configure Preferences	No Assignment
Sort File Tabs by Use	No Assignment
Indent with String	No Assignment
Maximize All	No Assignment
Minimize	No Assignment
Minimize All	No Assignment
Multiply	No Assignment
Clear All Bookmarks	No Assignment
Contact Information	No Assignment
Delete Blank Lines	No Assignment
Next Paragraph	No Assignment
Set Clipboard Previous	No Assignment
Convert Case Title	No Assignment

Convert Case Upper	No Assignment
Convert Case Words	No Assignment
ANSI to OEM	No Assignment
Run Macro 16	No Assignment
Copy Filename	No Assignment
Ctags Function Index	No Assignment
ASCII to EBCDIC	No Assignment
Open Program at Cursor	No Assignment
Run Macro 22	No Assignment
Open Recent File 2	No Assignment
Open Recent File 3	No Assignment
Convert Case Invert	No Assignment
Open Recent File 5	No Assignment
Convert Case Sentences	No Assignment
Open Recent File 7	No Assignment
Boxer Software Website	No Assignment
Open Recent File 9	No Assignment
Open Recent File 10	No Assignment
Open Recent File 11	No Assignment
Open Recent File 12	No Assignment
Open Recent File 13	No Assignment
Open Recent File 14	No Assignment
Open Recent File 15	No Assignment
Open Recent File 16	No Assignment
Open Recent File 17	No Assignment
Open Recent File 18	No Assignment
Open Recent File 19	No Assignment
Open Recent File 20	No Assignment
Open Recent File 21	No Assignment
Open Recent File 22	No Assignment
Open Recent File 23	No Assignment
Open Recent File 24	No Assignment
Open Recent Project 1	No Assignment

Open Recent Project 2	No Assignment
Open Recent Project 3	No Assignment
Open Recent Project 4	No Assignment
Open Recent Project 5	No Assignment
Switch to Window 6	No Assignment
Open Recent Project 7	No Assignment
Open Recent Project 8	No Assignment
Open Recent Project 9	No Assignment
Email Boxer Software	No Assignment
Open Recent File 4	No Assignment
Open Recent Project 12	No Assignment
Delete Bookmarked Lines	No Assignment
Configure Auto-Complete	No Assignment
Open Recent File 8	No Assignment
Configure Ctags Function Index	No Assignment
Open System Files	No Assignment
FAQs	No Assignment
Configure Printer Font	No Assignment
Configure Screen Font	No Assignment
Order with PayPal	No Assignment
Page Left	No Assignment
Page Right	No Assignment
Fill with String	No Assignment
Configure User Tools	No Assignment
About Boxer	No Assignment
Find and Count	No Assignment
Run Macro 10	No Assignment
Run Macro 11	No Assignment
Run Macro 12	No Assignment
Run Macro 13	No Assignment
Run Macro 14	No Assignment
Run Macro 15	No Assignment
Clear Clipboard 7	No Assignment

Run Macro 17	No Assignment
Run Macro 18	No Assignment
Find Unique Lines	No Assignment
Pause Recording	No Assignment
Open Recent File 1	No Assignment
User List 5	No Assignment
Auto-Number	No Assignment
Previous Paragraph	No Assignment
Print All Monochrome	No Assignment
Open Recent File 6	No Assignment
Print All Normal	No Assignment
Print Color	No Assignment
Print Monochrome	No Assignment
Bring User Lists to Top	No Assignment
Print Preview Color	No Assignment
Print Preview Monochrome	No Assignment
Print Preview Normal	No Assignment
Print Setup	No Assignment
Project Add All	No Assignment
Check for Latest Version	No Assignment
Project Auto-Update	No Assignment
Split Vertical	No Assignment
Project Delete	No Assignment
Project Edit Active	No Assignment
Project Edit Other	No Assignment
Project New	No Assignment
Project Open	No Assignment
Run Macro 44	No Assignment
Project Update All	No Assignment
Project Update One	No Assignment
Run Macro 47	No Assignment
Run Macro 48	No Assignment
Clear Recent Files List	No Assignment

Redo All	No Assignment
Reference	No Assignment
Clear Windows Clipboard	No Assignment
View Hex Ruler	No Assignment
Invert Lines	No Assignment
Close All but Active	No Assignment
Color Chart	No Assignment
Left Window Edge	No Assignment
Comment	No Assignment
Configure Colors	No Assignment
Restore All	No Assignment
Make Line Bottom	No Assignment
Right Window Edge	No Assignment
Run Macro 1	No Assignment
Run Macro 2	No Assignment
Run Macro 3	No Assignment
Run Macro 4	No Assignment
Run Macro 5	No Assignment
Run Macro 6	No Assignment
Run Macro 7	No Assignment
Run Macro 8	No Assignment
Run Macro 9	No Assignment
Total and Average	No Assignment
Uncomment	No Assignment
OEM Chart	No Assignment
OEM to ANSI	No Assignment
Undo All	No Assignment
Run Macro 46	No Assignment
User Tool 22	No Assignment
User Tool 23	No Assignment
User List 1	No Assignment
Run Macro 19	No Assignment
Run Macro 20	No Assignment

User List 4	No Assignment
Run Macro 37	No Assignment
Run Macro 23	No Assignment
Run Macro 24	No Assignment
Run Macro 25	No Assignment
Run Macro 26	No Assignment
Run Macro 27	No Assignment
Run Macro 28	No Assignment
Run Macro 29	No Assignment
Run Macro 30	No Assignment
Run Macro 31	No Assignment
Run Macro 32	No Assignment
Run Macro 33	No Assignment
Run Macro 34	No Assignment
Run Macro 35	No Assignment
Run Macro 36	No Assignment
Save a Copy As	No Assignment
Run Macro 38	No Assignment
Run Macro 39	No Assignment
Run Macro 40	No Assignment
Run Macro 41	No Assignment
User Tool 17	No Assignment
Run Macro 43	No Assignment
User Tool 19	No Assignment
Run Macro 45	No Assignment
User Tool 21	No Assignment
Explore Program Folder	No Assignment
Order Boxer	No Assignment
Run Macro 49	No Assignment
Run Macro 50	No Assignment
Edit Clipboard 6	No Assignment
Select Page Right	No Assignment
Word Count	No Assignment

Edit Windows Clipboard	No Assignment
Save Key Recording	No Assignment
Save Selection As	No Assignment
Open Recent Project 13	No Assignment
Error Chart	No Assignment
Restore	No Assignment
Open Recent Project 16	No Assignment
Explore Data Folder	No Assignment
Shaded Tab Zones	No Assignment
Sort File Tabs by Extension	No Assignment
Window List	No Assignment
Maximize	No Assignment
Select Page Left	No Assignment
User List 3	No Assignment
View File Tabs	No Assignment
Page Setup	No Assignment
Find a Disk File	No Assignment
User Tool 15	No Assignment
Strip HTML/XML Tags	No Assignment
User Tool 1	No Assignment
User Tool 18	No Assignment
User Tool 3	No Assignment
User Tool 20	No Assignment
Load Key Recording	No Assignment
View Toolbar	No Assignment
Sort File Tabs by Name	No Assignment
User Tool 24	No Assignment
User List 2	No Assignment
Spaces to Tabs	No Assignment
Project Add One	No Assignment
Split Horizontal	No Assignment
Go to Byte Offset	No Assignment
User List 7	No Assignment

Print All Color	No Assignment
User Tool 16	No Assignment
Set Clipboard Next	No Assignment
Tabs to Spaces	No Assignment
User Tool 4	No Assignment
Open Recent Project 14	No Assignment
Open Recent Project 15	No Assignment
Tile Across	No Assignment
User Tool 8	No Assignment
Sort Lines	No Assignment
Value at Cursor	No Assignment
Open Recent Project 6	No Assignment
Toolbar Bottom	No Assignment
User Tool 13	No Assignment
Insert Formfeed	No Assignment
Synchronized Scroll	No Assignment
Syntax Highlight As	No Assignment
View Status Bar	No Assignment
Open Recent Project 11	No Assignment
Technical Support	No Assignment
View Syntax Highlighting	No Assignment
Toolbar Left	No Assignment
User Tool 5	No Assignment
Tile Down	No Assignment
Make Line Top	No Assignment
User Tool 9	No Assignment
Toggle Read-Only	No Assignment
User Tool 10	No Assignment
Open Recent Project 10	No Assignment
View Horizontal Scroll Bar	No Assignment
Toolbar Top	No Assignment
User Tool 2	No Assignment
User List 8	No Assignment

View Vertical Scroll Bar	No Assignment
Toolbar Right	No Assignment
User Tool 6	No Assignment
User Tool 11	No Assignment
User Tool 14	No Assignment
Window Skip	No Assignment
User Tool 7	No Assignment
User Tool 12	No Assignment
User List 6	No Assignment
Page Down	PgDn
Page Up	PgUp
Right	Right
Paste Windows Clipboard	Shift+Alt+0
Paste Clipboard 1	Shift+Alt+1
Paste Clipboard 2	Shift+Alt+2
Paste Clipboard 3	Shift+Alt+3
Paste Clipboard 4	Shift+Alt+4
Paste Clipboard 5	Shift+Alt+5
Paste Clipboard 6	Shift+Alt+6
Paste Clipboard 7	Shift+Alt+7
Paste Clipboard 8	Shift+Alt+8
FTP Save As	Shift+Alt+F12
FTP Open	Shift+Alt+O
Hex Mode	Shift+Alt+X
Unindent	Shift+BkSp
Insert Symbol 1	Shift+Ctrl+1
Insert Symbol 2	Shift+Ctrl+2
Insert Symbol 3	Shift+Ctrl+3
Insert Symbol 4	Shift+Ctrl+4
Insert Symbol 5	Shift+Ctrl+5
Insert Symbol 6	Shift+Ctrl+6
Insert Symbol 7	Shift+Ctrl+7
Insert Symbol 8	Shift+Ctrl+8

Append	Shift+Ctrl+C
Next Bookmark	Shift+Ctrl+Down
Select to End of File	Shift+Ctrl+End
Insert Line Above	Shift+Ctrl+Enter
Flip Case	Shift+Ctrl+F
Insert Long Date	Shift+Ctrl+F11
Insert Long Time	Shift+Ctrl+F12
Go to Column	Shift+Ctrl+G
Select to Start of File	Shift+Ctrl+Home
Select Word Left	Shift+Ctrl+Left
Reload File	Shift+Ctrl+O
Select to Bottom of Page	Shift+Ctrl+PgDn
Select to Top of Page	Shift+Ctrl+PgUp
Replace Again	Shift+Ctrl+R
Select Word Right	Shift+Ctrl+Right
Save All	Shift+Ctrl+S
Previous Bookmark	Shift+Ctrl+Up
Paste As	Shift+Ctrl+V
Cut Append	Shift+Ctrl+X
Cut	Shift+Del
Select Down	Shift+Down
Select to End of Line	Shift+End
Help On	Shift+F1
Open Context Menu	Shift+F10
Insert Short Date	Shift+F11
Insert Short Time	Shift+F12
Duplicate & Increment	Shift+F2
Find Previous	Shift+F3
Swap Words	Shift+F4
Record Keys	Shift+F5
Window Previous	Shift+F6
Check Word	Shift+F7
Bookmark Manager	Shift+F9

Select to Start of Line	Shift+Home
Paste	Shift+Ins
Select Left	Shift+Left
Select Page Down	Shift+PgDn
Select Page Up	Shift+PgUp
Select Right	Shift+Right
Indent one Tabstop	Shift+Tab
Select Up	Shift+Up
Insert Tab	Tab
Up	Up

6.7 Dropping Text Files onto Boxer

Dragging and Dropping Files refers to the process of selecting one or more file icons, dragging them to a destination program, and releasing the left mouse button. A single file icon can be selected by clicking on it with the left mouse button. Additional files can be added to the selection by depressing *Ctrl* while clicking with the left mouse button. A range of file icons can be selected by selecting the first icon in the range, and depressing *Shift* before selecting the last icon in the range.

While Boxer is active, it recognizes the dropping of files onto its window as a request to add the file(s) to the existing editing session.

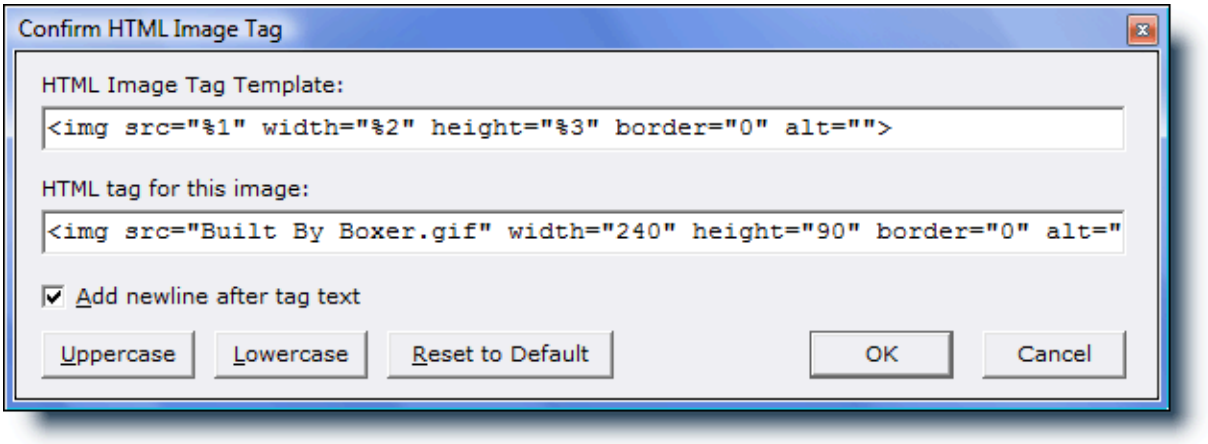
When Boxer is inactive, dropping one or more files onto its program icon will cause a new edit session to be launched. If files are dropped onto the program icon while an editing session is underway, these files will be added to the existing session.

When a Project file is dropped onto Boxer, the files named within the Project file will be opened. See the New Project topic for more details on this capability.

6.8 Dropping Image Files onto Boxer

Boxer will react to a dropped image file by creating an HTML image tag declaration in the current file, at the location of the text cursor. The image tag will use the filename, image height, and image width of the dropped image file. The following image file formats are supported: BMP, GIF and JPEG.

Before the image tag is created, a dialog appears to confirm the operation, and to provide access to the image template:



You can control the format of the image tag by editing the *HTML Image Tag Template* in the upper edit box. The format of the template string can be changed freely, so long as the `%1`, `%2` and `%3` sequences appear in the string, and remain associated with the filename (`src`), `width` and `height` properties, respectively. The image tag that will be inserted appears in the lower edit box. Buttons are provided to quickly convert the tag to uppercase or lowercase, as well as a *Reset to Default* button that will restore the template string to its original form.

This function is also accessible by issuing the HTML Image Tag command from the main menu.

6.9 File Associations

Windows provides the ability to associate a file type (file extension) with the application program which is used to open that file. This relationship is termed a *file association*. Microsoft Word establishes a file association so that when a `.DOC` file is double-clicked, Word will be used to open that file. Likewise, your Internet browser has established an association between itself and files with an `HTML` extension.

You might wish to associate one or more file types with Boxer so that it will be launched whenever one of these file types is double-clicked from within Explorer. Some users prefer this method of opening or loading files to using the Open command from within Boxer to open files.

During installation, Boxer's Setup program provided the opportunity to associate several common text file extensions with Boxer. If you elected to do this, some file types will already be associated with Boxer. In this case you will not need to establish associations for these file types: `.TXT` files, `.INI` files, `.BAT` files and `.BTM` files.

The method by which a file association is established depends on whether or not that file type is already associated with another application.

When the File Type is Not Yet Associated
When you double-click on a file whose type has not yet been associated with another program, Windows presents a dialog box that allows you to select the program which

should be used to open files of that type. Boxer should appear in the list of programs which can be used to open the file. If it does not, use the *Other...* button to locate the Boxer program. This dialog box contains a checkbox titled *Always use this program to open this file.* If you select this option, Windows will remember the association and Boxer will automatically be used to open files of this type in the future. By supplying a short description in the box provided, the file type will become registered using the description provided.

When the File Type is Already Associated
If the file type you would like to associate with Boxer has already been associated with another program, a different method must be used to establish the new association. Double-clicking on the file will simply launch the associated program, and no opportunity will be provided to specify the program to be launched as occurs for an unassociated file type.

The first step is to remove the existing file association. From within Explorer, select the *Options* command in the *View* menu (in Windows 98 this command is titled *Folder Options*). Click the *File Types* tab in the *Options* dialog box. A list of registered file types is displayed. Find the file type which is to be changed within the list of registered types. Check the information in the *File type details* display to be sure the right file type has been located. When you are **sure** you have located the correct file type, proceed as follows:

Click the *Edit...* button in the *Options* dialog box
the *Edit File Type* dialog box appears

Click the **open** action in the *Actions* listbox

Click the *Edit...* button in the *Edit File Type* dialog box
the *Editing Action for Type:* dialog box appears

In the edit box titled *Application used to perform action*, enter the full filepath to the Boxer program. Use the *Browse...* button if needed to locate file.

Make sure that following the filepath is a Space character and the characters `"%1"`, including the double quotes. This is very important.

Click *OK* in the *Editing Action for Type:* dialog box

Click *Close* in the *Edit File Type* dialog box appears

Click *Close* in the *Options* dialog box appears

The file association has been changed. Repeat this process as required to create other file associations.

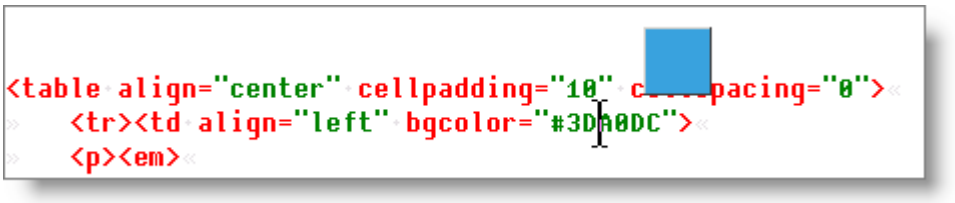

In the most recent versions of Windows, it is no longer possible to establish a file association between an application (such as Boxer) and a file that has no file extension. Such associations had been permitted in earlier versions of Windows, but this appears to have changed in Windows Me, 2000, XP etc.

When creating a file association to edit a Batch file, be sure to associate Boxer with the **edit** action, and not the **open** action. 'Opening' a batch file has special meaning, as compared to other text files, because the command processor is charged with interpreting and processing the commands in the batch file.

You might wish to associate Boxer with the **print** action for some file types to gain quicker access to Boxer's Color Syntax Printing capability. Once this association is made, right-clicking on a file of the defined type and selecting the *Print* option from its context menu will launch Boxer so the file can be printed.

6.10 HTML Color Code Popup Hints

Boxer will display a popup color hint when the mouse cursor is allowed to hover over an HTML color code, such as "#3D121F", or "DarkSlateBlue".



This feature can be very helpful when editing HTML code, as it provides a means to check colors visually without the need to refer to a chart.

Color codes of the following forms are recognized:

```
=red
="red"
= "red"
=#FF0000
="#FF0000"
= "#FF0000"
rgb(255,0,0)
```

More than 150 official HTML color code names are recognized.

This feature can be disabled with a checkbox on the Configure | Preferences | Display dialog page.

To select colors and insert HTML color codes, use the HTML Color Chart command.

6.11 Insert Symbols

Default Shortcut Key: Shift+Ctrl+1 to Shift+Ctrl+8

The Insert Symbols feature provides a mechanism for defining up to eight character values which can be entered into a text file using an associated key assignment. This feature is useful for defining characters which are not readily insertable from the

keyboard, such as accented characters or special symbols which reside in the upper half of the ASCII character set.

The symbol values are defined on the Configure | Preferences | Editing 2 options page. The default key assignments used for inserting the eight symbols are *Shift+Ctrl+1* to *Shift+Ctrl+8*. After defining the symbols you prefer you might wish to use the Configure | Keyboard command to associate mnemonic assignments with each symbol, such as *Shift+Ctrl+A* to insert an accented letter 'a'.

6.12 Inserting Special Characters

For purposes of this discussion, special characters are those characters that are not readily insertable from a keyboard using the standard keys provided. For many users these characters are the Greek characters, math symbols, frame drawing characters and accented characters. The ANSI Chart and OEM Chart commands can be used to view the characters available in each character set, along with their associated character values.

While special characters cannot be inserted directly from the keyboard using a single key, there is a technique that enables these characters to be inserted. The technique involves typing the character's value from the Numeric keypad while the *Alt* key is depressed. The method is slightly different depending upon whether ANSI or OEM character values are being used:

ANSI Characters
To insert a special character by entering its ANSI character value, do the following:

1. Depress and hold the *Alt* key
2. Type a zero from the Numeric keypad
3. Type the digits that define the character value from the Numeric keypad
4. Release the *Alt* key

OEM Characters
To insert a special character by entering its OEM character value, do the following:

1. Depress and hold the *Alt* key
2. Type the digits that define the character value from the Numeric keypad
3. Release the *Alt* key

Notice that ANSI character values have the form `0###` while OEM character values lack the leading zero: `###`

☞ The ANSI Chart and OEM Chart permit characters to be inserted by double-clicking or pressing *Enter*.

☞ A special character can also be inserted by supplying its value to the Insert Character command.

☞ The Insert Symbols feature permits up to eight special characters to be defined for easier input. Once defined, these symbols can be entered with a single key

sequence.

☞ The standard Windows controls (editbox, combobox, etc) do not normally permit most low ASCII values (below 32) to be entered, even using the special technique described above. Boxer has taken special care to defeat this default behavior so that the following dialogs can accept low ASCII values for input: Find, Replace, Replace Line Enders, Find and Count, Find Text in Disk Files and Fill with String.

☞ If an OEM character value is typed into the editor while an ANSI font is in use, an automatic conversion takes place. Example: In the OEM character set, the value of the ¼ character is 172. In the ANSI character set, this character has the value 188. But if the value 172 is entered using the technique described above, while an ANSI font is in use, character value 188 is inserted. The converse will occur if an ANSI character value is inserted while an OEM font is in use.

6.13 Installing or Reinstalling Boxer

Installing from CD

To install or reinstall Boxer, locate your Boxer CD and place it into your CD drive. Ordinarily, the setup program will run automatically and you will be guided through the quick installation. If the setup program does not run automatically, do the following: from the Start Menu, choose Run, and then type `J:\Setup.exe`, where 'J' represents the drive letter of your CD-ROM drive. Follow the on-screen instructions which are presented by the setup program.

Installing from a Downloaded File

To install or reinstall Boxer, execute the file which was downloaded in order to run Boxer's setup program. Follow the on-screen instructions which are presented by the setup program.

Installing from Diskette

To install or reinstall Boxer, locate your Boxer diskette and place Disk 1 into your floppy drive. From the Start Menu, choose Run, and then type `A:\Setup.exe`. Follow the on-screen instructions which are presented by the setup program. When prompted, insert Disk 2 into the floppy drive.

Reinstalling atop a Previous Version

If you are reinstalling, Boxer's setup program will automatically sense the existing installation and suggest the directory in which Boxer is installed as the location for the new installation. Unless you have a compelling reason not to do so, you should accept this location.

- Boxer will not overwrite any data files you may have created in its *data folder* or *program folder*, or anywhere else.
- Boxer will not overwrite the User List files from a prior installation, since you may

have created custom lists.

- Boxer will not overwrite the Auto-Complete dictionary file `AC_words.txt`, since you might have edited it.
- Boxer will not overwrite the `Template.ini` file, since you may have defined custom Templates.
- Boxer will not overwrite the Syntax Highlighting information file named `Syntax.ini`, since you may have added new language information to it. Boxer will install a new copy of `Syntax.ini` using the name `New Syntax.ini`, so that you will have access to any new language information which may have been added since your version was released. In this case you will need to manually copy the new information from `New Syntax.ini` into `Syntax.ini` so that your custom additions in `Syntax.ini` will be preserved.

6.14 Intellimouse Support

Boxer supports the use of the Microsoft Intellimouse pointing device. The Intellimouse features a *mousewheel* which can be used to scroll the document within the current window. The amount of scroll which results from a single turn of the wheel can be configured from the device settings for the Intellimouse.

The mousewheel behaves like a center mouse button when pressed. In Boxer, this means that the mousewheel can be used to initiate columnar text selection.

By default, support for the Intellimouse is enabled. Support can be disabled from the Configure | Preferences | Other options page.

6.15 Macro Examples

The following example macros show the syntax of Boxer's macro language, while also suggesting useful methods of attack for common programming tasks:

Move cursor to bottom of paragraph

```
// move the cursor to the bottom line of the current paragraph

macro BottomOfParagraph()
{
while (LineNumber < LineCount && !LineIsEmpty(LineNumber+1))
   Down;
}
```

Move cursor to top of previous paragraph

```
// move the cursor to the top line of the previous paragraph

macro TopOfPreviousParagraph()
{
Up;

while (LineNumber > 1 && !LineIsEmpty(LineNumber-1))
   Up;

StartOfLine;
}
```

Move cursor to top of current paragraph

```
// move the cursor to the top line of the current paragraph

macro TopOfCurrentParagraph()
{
while (LineNumber > 1 && !LineIsEmpty(LineNumber-1))
   Up;
}
```

Move cursor to top of next paragraph

```
// move the cursor to the first line of the next paragraph

macro TopOfNextParagraph()
{
while (LineNumber < LineCount && !LineIsEmpty(LineNumber))
   Down;

Down;
StartOfLine;
}
```

Add a newline after every closing angle bracket

```
// add a newline after each closing angle (>) character
// unless the angle already appears at end of line
```

```
macro AddNewlineAfterCloseAngle()
{
int line, i, j;
string str;

// loop on all lines in the file
for (line = 1; line <= LineCount(); line++)
   {
   // get the text of line 'line' into string 'str'
   GetLineText(line, str);

   // get the index of the closing angle
   j = strchr(str, '>');

   // if the character was found and was not at end-of-line...
   if (j != -1 && str[j+1] != '\0')
      {
      GotoLine(line);
      GotoColumn(1);

     // advance the cursor to the character
      while (ValueAtCursor() != '>')
         Right;

      // and past the character
      Right;

      // insert a newline
      Enter;

      // process this line again in case other tags exist
      line--;
      }
   }
}
```

Apply HTML markup to a simple text file

```
// apply HTML markup to a simple text file
// also converts double quote, ampersand, and
// angle brackets to HTML equivalents

macro ApplyHTMLMarkup()
{
```

```
int prevlen, len, i;
string str;
int numchanges;

// loop on all lines in the file
for (i = 1; i <= LineCount; i++)
   {
   // get the text of line 'i' into 'str'
   GetLineText(i, str);

   // reset the change counter
   numchanges = 0;

   // convert sensitive characters to HTML codes
   numchanges += ChangeString(str, "&",  "&amp;");
   numchanges += ChangeString(str, "<",  "&lt;");
   numchanges += ChangeString(str, ">",  "&gt;");
   numchanges += ChangeString(str, "\"", "&quot;");

   // if changes were made, replace the line's text
   if (numchanges > 0)
      PutLineText(i, str);
   }

// move to top of file
StartOfFile;
Down;

// loop on all lines in the file, starting on line 2
for (i = 2; i <= LineCount; i++)
   {
   // get the length of the previous line
   prevlen = LineLength(i-1);

   // get the length of this line
   len = LineLength(i);

   // if this line is empty, and the previous line isn't...
   // apply <br> markers to the end of the line
   if (len == 0 && prevlen != 0)
      {
      Up;
      EndOfLine;
      PutString("<br><br>");
      StartofLine;
      Down;
```

```
    }

  Down;         // move down to the next line
  }

StartOfFile;

PutString("<html>\n");
PutString("<head>\n");
PutString("<title></title>\n");
PutString("</head>\n\n");
PutString("<body>\n");

EndOfFile;
EndOfLine;
PutString("\n");
PutString("</body>\n");
PutString("</html>\n");

// place cursor between title and /title
GotoLine(3);
GotoColumn(8);
}
```

Display an ASCII chart in a new file

```
// ASCII chart example

macro ASCIIchart(void)
{
char i;

// open a new file
New;

// loop from space to 255 to show all chars
for (i = ' '; i <= 255; i++)
   printf("The ASCII value of '%c' is %d\n", i, i);
}
```

Convert comma-separated-value (CSV) data

```
// convert comma-separated-value (CSV) data on the current
```

```
// line so that each field is placed on its own line

macro ConvertCSV()
{
string str;
int numquotes, numcommas;

// get the count of quotes/commas on this line
numquotes = LineContains(linenumber, "\"");

numcommas = LineContains(linenumber, ",");

// if this appears to be CSV data...
if (numcommas+1 == numquotes / 2)
   {
   // get the text of the current line
   GetLineText(linenumber, str);

   // remove any empty data fields
   ChangeString(str, "\"\",", "");

   // convert "," to a newline
   ChangeString(str, "\",\"", "\n");

   // remove the first and last quotes
   ChangeString(str, "\"", "");

   // select the line
   GoToColumn(1);
   SelectToEndOfLine;

   // replace the selection
   PutString(str);
   }

// position for next line
Down;
StartOfLine;
}
```

Cut lines containing a user-defined string

```
// cut lines containing a user-defined string to the Windows clipboard
```

```
macro CutLinesContaining();
{
int line;
int len;
string str;
int numcut = 0;

// get the string from the user
len = GetString("Cut lines containing this string:", str);

if (len == 0)
   return;

// make the Windows clipboard the active clipboard
SetClipboard(0);

// clear the Windows clipboard
ClearClipboard(0);

// move cursor to start of file
StartOfFile();

// loop on all lines in the file
for (line = 1; line <= LineCount(); line++)
   {
   // does this line contain the string?
   if (LineContains(line, str))
      {
      GotoLine(line);
      CutAppend();
      numcut++;          // tally the cut
      line--;            // stay here for next line
      }
   }

// report the results
if (numcut == 1)
   message("Results", "1 line was cut to the Windows clipboard");
else
   message("Results", numcut,
           " lines were cut to the Windows clipboard");
}
```

Delete blank lines

```
// delete blank lines in the current file

macro DeleteBlankLines(void)
{
int i, len;

// start at the top of the file
StartOfFile;

// loop on all lines in the file
for (i = 1; i <= LineCount; i++)
   {
   // get the length of this line
   len = LineLength(i);

   // is this line empty?
   if (len == 0)
      {
      DeleteLine;     // delete this line
      i--;            // stay at this line #
      }
   else
      {
      Down;           // move down to the next line
      }
   }
}
```

Delete lines containing a user-defined string

```
// deletes lines containing a user-defined string

macro DeleteLinesContaining()
{
int line;
int len;
string str;
int deleted = 0;

// get the string from the user
len = GetString("Delete lines containing this string:", str);
```

```
if (len == 0)
   return;

// loop on all lines in the file
for (line = 1; line <= LineCount(); line++)
   {
   // does this line contain the string?
   if (LineContains(line, str))
      {
      DeleteLine(line);   // delete it
      deleted++;          // tally the deletion
      line--;             // stay here for next line
      }
   }

// report the results
if (deleted == 1)
   message("Results", "1 line was deleted");
else
   message("Results", deleted, " lines were deleted");
}
```

Delete lines NOT containing a user-defined string

```
// deletes lines NOT containing a user-defined string

macro DeleteLinesNotContaining()
{
int line;
int len;
string str;
int deleted = 0;

// get the string from the user
len = GetString("Delete lines that do NOT contain this string:", str);

if (len == 0)
   return;

// loop on all lines in the file
for (line = 1; line <= LineCount(); line++)
   {
   // does this line contain the string?
   if (!LineContains(line, str))
```

```
        {
        DeleteLine(line);   // delete it
        deleted++;          // tally the deletion
        line--;             // stay here for next line
        }
    }

// report the results
if (deleted == 1)
    message("Results", "1 line was deleted");
else
    message("Results", deleted, " lines were deleted");
}
```

Compute return on a deposited amount

```
// Compute result of amount left on deposit with continuous
// compounding.  Uses the formula: P = pe^rt

macro ComputeDeposit()
{
float amt, newamt, rate, years;
string str;

GetFloat("Enter the amount on deposit:", amt);

GetFloat("Enter the interest rate:", rate);

// if user entered 5, make it .05, for example
if (rate > 1.0)
    rate /= 100.0;

GetFloat("Enter the number of years on deposit:", years);

newamt = amt * pow(e, rate * years);

sprintf(str, "The amount with interest applied is: %.2f", newamt);
Message("Result", str);
}
```

Add blank lines after lines ending with !.?

```
// add a blank line after any line that ends with !.?

macro AddBlankLines(void)
{
char ch;
int i;

// loop on all lines in the file
for (i = 1; i <= LineCount; i++)
   {
   // make sure this line is not empty
   if (LineLength(i) > 1)
      {
      // move the cursor to this line
      GotoLine(i);

      // move to the end of the line
      EndOfLine;

      // backup off newline and onto last char
      Left;

      // get the value of char at the cursor
      ch = ValueAtCursor();

      // if it's a line ender, add Enter
      if (ch == '.' || ch == '?' || ch == '!')
         {
         EndOfLine;
         Enter;
         }
      }
   }
}
```

Double space and reformat

```
// double space and reformat the text on the clipboard
// prepares a web document for printing

macro DoubleSpaceAndReformat(void)
{
int i;
int numlines;
```

```
// save various editor settings
SaveSettings;

// open a new file and paste from clipboard
New;
Paste;

// set Text Width to 96
TextWidth(96);

// delete all blank lines
DeleteBlankLines;

// record the number of lines BEFORE we start adding lines
numlines = LineCount - 1;

// go to the top
StartOfFile;

// double space the file
for (i = 1; i <= numlines; i++)
   {
   Down;
   PutString("\n");
   }

// reformat the whole file
SelectAllText;
Reformat;

// remove any small indents that might be present
SelectAllText;
for (i = 1; i <= 12; i++)
Unindent;

// release the selection
Deselect;

// restore various editor settings
RestoreSettings;
}
```

Extract email addresses

```
// extract email addresses from all lines within the current file
// and append them to the end of the file

macro ExtractEmailAddresses()
{
int i, line, origlinecount;
int inword, isdelim;
int atsigns, dots;
int startword, endword;
string linetext, email;
char c;

// note the linecount before we start adding more lines
origlinecount = LineCount;

// loop on all lines in the file
for (line = 1; line <= origlinecount; line++)
   {
   // get the text of the whole line into the string 'linetext'
   GetLineText(line, linetext);

   // add a space to make end-of-line handling smoother
   strcat(linetext, " ");

   // loop on all characters in 'linetext'
   for (inword = FALSE, i = 0; linetext[i] != 0; i++)
      {
      c = linetext[i];

      // set a flag if this character is one that delimits words
      if (isalnum(c) || (strchr("_@.-", c) != -1))
         isdelim = FALSE;
      else
         isdelim = TRUE;

      // decide whether this character starts a new word,
      // or ends an existing word
      if (inword && isdelim)
         {
         inword = FALSE;
         endword = i-1;

         // we've just left a word: see if it had both
         // the required characters
         if (atsigns == 1 && dots >= 1)
            {
```

```
            // get the linetext address into a string
            SubString(email, linetext, startword, endword-startword+1);

            // add it to the end of the file
            EndOfFile;
            EndOfLine;
            Enter;
            PutString(email);
            }
         }
      else if (!inword && !isdelim)
         {
         inword = TRUE;
         startword = i;
         atsigns = 0;
         dots = 0;
         }

      // tally whether or not we see the required chars while we're in a
      if (inword)
         {
         if (linetext[i] == '@')
            atsigns++;

         if (linetext[i] == '.')
               dots++;
         }
      }
   }
}
```

Hex to Decimal

```
// shows hex to decimal conversion technique

macro HexToDecimal()
{
string str;
int x = 0;
int i, val;
char ch;

GetString("Enter a hexadecimal string", str);
```

```
for (i = 0; str[i] != '\0'; i++)
   {
   ch = str[i];

   if (!isxdigit(ch))
      {
      message("Error", "Invalid character encountered: ", ch);
      return;
      }

   ch = toupper(ch);

   if (isalpha(ch))
      val = ch - 'A' + 10;
   else
      val = ch - '0';

   x = x * 16;
   x = x + val;
   }

message("Result", "The decimal value is ", x);
}
```

Obfuscate the selected text with HTML codes

```
// convert the word selected into its HTML coded format

// this can be used to convert phone numbers and email addresses
// in web pages to frustrate automated crawlers from harvesting
// your information for spam lists

macro Obfuscate()
{
int i;
string str;
string result;
string tmp;

if (!TextIsSelected)
   {
   Message("Error",
      "Please select a word before\nrunning the macro.\n");
   return;
```

```
    }

GetSelection(str);

// loop on all characters in 'str'
for (i = 0; str[i] != '\0'; i++)
    {
    sprintf(tmp, "&#%03d;", str[i]);
    strcat(result, tmp);
    }

PutSelection(result);
}
```

Display 24-hour time

```
// display the current time in 24-hour time format

macro Print24HourTime()
{
int h, m, s;

GetTime24(h, m, s);
printf("%d:%02d:%02d", h, m, s);
}
```

Reduce blank lines

```
// reduce multiple blank lines to one blank line

macro ReduceBlankLines(void)
{
int thislen, prevlen, i;

// position cursor to line 2
StartOfFile;
Down;

// loop on all lines in the file (starting with line 2)
for (i = 2; i <= LineCount; i++)
    {
    // get the length of the previous line
    prevlen = LineLength(i-1);
```

```
    // get the length of this line
    thislen = LineLength(i);

    // are both previous and this line empty?
    if (prevlen == 0 && thislen == 0)
        {
        DeleteLine;     // delete this line
        i--;            // stay at this line #
        }
    else
        {
        Down;           // move down to the next line
        }
    }
}
```

Reformat to an alternative text width

```
// Reformat the current paragraph to 70 characters, regardless
// of what the current Text Width setting is

macro ReformatAlternative()
{
SaveSettings;
TextWidth(70);
Reformat;
RestoreSettings;
}
```

Extract double quoted strings

```
// extract double quoted strings from the current file
// and append them at the bottom of the file

macro ExtractStrings()
{
string s, s1, s2, s3;
int i, j, k;
int found = 0;
int original_linecount = LineCount();
```

```
// loop on all lines in the current file
for (i = 1; i <= original_linecount; i++)
   {
   // does this line have two or more double quotes?
   while (LineContains(i, "\"") >= 2)
      {
      // tally number of strings found
      found++;

      // get the text of line 'i' into string 's'
     GetLineText(i, s);

      // get the offset of the first double quote
      j = strchr(s, '\"');

      // get the index of the second double quote
      for (k = j + 1; s[k] != '\"'; k++)
         ;

      // get the first portion into 's1'
      SubString(s1, s, 0, j);

      // get the second portion (the string) into 's2'
     SubString(s2, s, j, k-j+1);

      // get the third portion into 's3'
     SubString(s3, s, k+1, 2048);

      // build the new line and replace it
      s = s1;
      s += s3;
     PutLineText(i, s);

      // gather the strings at the bottom of the current file
     EndOfFile();
     EndOfLine();
     printf("\n%s", s2);
      }
   }

// report the results
if (found == 1)
   printf("\n\n%d string was found and removed\n", found);
else
   printf("\n\n%d strings were found and removed\n", found);
}
```

Reverse the text of each line

```
// reverse the text on every line in the file
// "abcdefg" becomes "gfedcba"

macro ReverseLineText()
{
int i, len, line;
string str;
char tmp;

// loop on all lines in the file
for (line = 1; line <= LineCount; line++)
   {
   len = linelength(line);

   // ignore lines too short/long
   if (len >= 2 && len < 2000)
      {
      // get the text of line 'line' into string 'str'
     GetLineText(line, str);

      // loop through half this line
     for (i = 0; i < len/2; i++)
         {
         // swap the characters...
         tmp = str[i];
         str[i] = str[len-1-i];
         str[len-1-i] = tmp;
         }

      // replace line with reversed line
      PutLineText(line, str);
      }
   }
}
```

Reverse names: Smith.Bob to Bob.Smith

```
// changes a list of "Smith.Bob" entries to "Bob.Smith"
```

```
macro ReverseNames()
{
int line, i;
string str;
string first, last;
string newstring;

// loop on all lines in the file
for (line = 1; line <= LineCount; line++)
   {
   // get the text of line 'line' into string 'str'
   GetLineText(line, str);

   // look for a '.' within 'str'
   i = strstr(str, ".");

   if (i != -1)
      {
      // 'last' gets 'i' chars from 'str' starting at index 0
      SubString(last,  str, 0,   i);

      // 'first' gets up to 100 chars from 'str' starting at index i+1
      SubString(first, str, i+1, 100);

      // build a new string from 'first' and 'last'
      sprintf(newstring, "%s.%s", first, last);

      // replace the text of the line
      PutLineText(line, newstring);
      }
   }
}
```

Truncate lines after a user-defined string

```
// truncate lines after a user-defined string

macro TruncateLineAfterString()
{
int j, line, len;
int truncated = 0;
string str, linestr, newstr;

// get the string from the user
```

```
len = GetString("Truncate lines after this string:", str);

// if the string is empty, quit
if (len == 0)
        return;

// loop on all lines in the file
for (line = 1; line <= LineCount; line++)
   {
   GetLineText(line, linestr);

   // does this line contain the string?
   if ((j = strstr(linestr, str)) != -1)
      {
      // create a new string without the trailing text
      SubString(newstr, linestr, 0, j+len);

      // replace this line with the new text
      PutLineText(line, newstr);

      // tally the truncation
      truncated++;
      }
   }

// report the results
if (truncated == 1)
   message("Results", "1 line was truncated");
else
   message("Results", truncated, " lines were truncated");
}
```

Truncate lines at a user-defined string

```
// truncate lines at a user-defined string

macro TruncateLineAtString()
{
int j, line, len;
int truncated = 0;
string str, linestr, newstr;

// get the string from the user
len = GetString("Truncate lines at this string:", str);
```

```
// if the string is empty, quit
if (len == 0)
        return;

// loop on all lines in the file
for (line = 1; line <= LineCount; line++)
   {
   GetLineText(line, linestr);

   // does this line contain the string?
   if ((j = strstr(linestr, str)) != -1)
      {
      // create a new string without the trailing text
      SubString(newstr, linestr, 0, j);

      // replace this line with the new text
      PutLineText(line, newstr);

      // tally the truncation
      truncated++;
      }
   }

// report the results
if (truncated == 1)
   message("Results", "1 line was truncated");
else
   message("Results", truncated, " lines were truncated");
}
```

Delete lines that begin with a user-defined string

```
// deletes lines that begin with a user-defined string

macro DeleteLinesThatBeginWith()
{
int line, len;
string str, linestr;
int deleted = 0;

// get the string from the user
len = GetString("Delete lines that begin with:", str);
```

```
if (len == 0)
   return;

// loop on all lines in the file
for (line = 1; line <= LineCount(); line++)
   {
         // get the text of line 'line' into 'linestr'
         GetLineText(line, linestr);
            // does this line start with 'str'?
         if (strncmp(linestr, str, len) == 0)
               {
               DeleteLine(line);   // delete it
               deleted++;          // tally the deletion
               line--;             // stay here for next line
               }
         }

// report the results
if (deleted == 1)
   message("Results", "1 line was deleted");
else
   message("Results", deleted, " lines were deleted");
}
```

Delete lines that end with a user-defined string

```
// deletes lines that end with a user-defined string

macro DeleteLinesThatEndWith()
{
int line, len, linelen;
string str, linestr, str2;
int deleted = 0;

// get the string from the user
len = GetString("Delete lines that end with:", str);

if (len == 0)
   return;

// loop on all lines in the file
for (line = 1; line <= LineCount(); line++)
   {
   // get the text of line 'line' into 'linestr'
```

```
    linelen = GetLineText(line, linestr);

    // if the line is too short, do nothing
    if (linelen < len)
       {
       ;
       }
    // does this line end with 'str' ?
    else
       {
       // isolate the tail of the line into a string
       SubString(str2, linestr, linelen - len, len);

       if (strcmp(str2, str) == 0)
          {
          DeleteLine(line);   // delete it
          deleted++;          // tally the deletion
          line--;             // stay here for next line
          }
       }
    }

// report the results
if (deleted == 1)
   message("Results", "1 line was deleted");
else
   message("Results", deleted, " lines were deleted");
}
```

Call MapQuest to show a map

```
// get an address from the user and call it up on MapQuest

macro CallMapQuest()
{
string city, state, address, country, url;
int zoom = 7;

// get information from the user
GetString("Enter street address:", address);
GetString("Enter city/town:",      city);
GetString("Enter state/province:", state);
GetString("Enter country:",        country);
```

```
// state level maps are better at zoom level 3
if (city == "")
   zoom = 3;

// country level maps are better at zoom level 1
if (state == "")
   zoom = 1;

// convert embedded spaces to plus signs
ChangeString(address, " ", "+");
ChangeString(city," ", "+");
ChangeString(state,   " ", "+");
ChangeString(country, " ", "+");

// build the URL that will be used
sprintf(url,
"http://www.mapquest.com/maps/map.adp?city=%s&state=%s&address=%s&c
ountry=%s&zoom=%d"    , city, state, address, country, zoom);

// send the URL to Windows so the default browser is run
OpenURL(url);
}
```

Convert British English punctuation to American English punctuation

```
// Convert British English punctuation to American English punctuation
// (in Britain, periods and commas are placed outside double quotes)

macro BritishPunctuation()
{
// notice that the double quote character must be escaped with a
// backslash when it appears within a string

// change ".  to  ."
ReplaceAll("\".",  ".\"");

// change ",  to  ,"
ReplaceAll("\"\,", ",\"");
}
```

6.16 Macro Function Reference

Function	Prototype and Description
abs()	int abs(int n) Returns the absolute value of 'n'.
acos()	float acos(float x) Returns the arc cosine of 'x' in radians. 'x' must be in the range -1 to 1.
ActiveClipboard	int ActiveClipboard Returns the number of the active clipboard. The Windows Clipboard is number 0; private clipboards are numbered 1 to 8.
ActiveSpellChecking()	int ActiveSpellCheck(int mode) Enables or disables Active Spell Checking according to 'mode'.
AlignCenter	Issues the Align Center command
AlignLeft	Issues the Align Left command
AlignRight	Issues the Align Right command
AlignSmooth	Issues the Align Smooth command
ANSIChart	Issues the ANSI Chart command
ANSItoOEM	Issues the ANSI to OEM command
Append	Append Append the selected text to the current clipboard. If no text is selected, the current line is appended to the clipboard.
AppendToClipboard()	int AppendToClipboard(string str, int n) Appends string 'str' to Clipboard 'n'. Returns the total length of the text on the clipboard, or -1 for error. The Windows Clipboard is number 0; private clipboards are numbered 1 to 8. See also PutClipboardText().
ApplyHighlighting	Issues the Apply Highlighting command
ArrangeIcons	Issues the Arrange Icons command
ASCIItoEBCDIC	Issues the ASCII to EBCDIC command
asin()	float asin(float x) Returns the arc sine of 'x' in radians. 'x' must be in the range -1 to 1.
atan()	float atan(float x) Returns the arc tangent of 'x' in radians.

atof()	float atof(string str) Returns the floating point value of the number described by string 'str'.
atoi()	int atoi(string str) Returns the integer value of the decimal number described by string 'str'.
AutoNumber	Issues the Auto-Number command
Backspace	Issues the Backspace command
Backtab	Issues the Backtab command
Beep()	Beep(int freq, int duration) Makes a sound through the PC speaker using the supplied values for frequency and duration. Frequency is in Hz and duration is in milliseconds. Beep(1000, 300) produces a standard beep.
BookmarkManager	Issues the Bookmark Manager command
BottomOfPage	Issues the Bottom of Page command
BringUserListsToTop	Issues the Bring User Lists to Top command
BrowseForFilename()	BrowseForFilename(string fn, int mustexist) Browse for a filename using a standard Windows open dialog and place the selected filename in 'fn'. If 'mustexist' is non-zero, the selected filename must already exist. If 'mustexist' is 0, a new filename can be selected. Returns 1 for success or -1 for error.
ByteCount	int ByteCount Returns the number of characters in the current file.
Calculator	Issues the Calculator command
Calendar	Issues the Calendar command
Cascade	Issues the Cascade command
CascadeHorizontal	Issues the Cascade Horizontal command
CascadeVertical	Issues the Cascade Vertical command
CaseInvert	Issues the Case Invert command
CaseLower	Issues the Case Lower command
CaseSentences	Issues the Case Sentences command
CaseTitle	Issues the Case Title command
CaseUpper	Issues the Case Upper command
CaseWords	Issues the Case Words command
ceil()	float ceil(float x) Returns (as a float) the smallest integer not less than 'x'. Example: ceil(1.5) returns 2.0.

ChangeString()	int ChangeString(string str1, str2, str3) Searches 'str1' and changes all occurrences of 'str2' to the string 'str3'. Returns the number of changes made or -1 for error. The search is case sensitive. Regular expressions are not recognized. If 'str3' is an empty string, the effect will be to delete all occurrences of 'str2' within 'str1'.
ChangeStringi()	int ChangeStringi(string str1, str2, str3) Searches 'str1' and changes all occurrences of 'str2' to the string 'str3'. Returns the number of changes made or -1 for error. The search is case insensitive. Regular expressions are not recognized. If 'str3' is an empty string, the effect will be to delete all occurrences of 'str2' within 'str1'.
ChangeStringRE()	int ChangeStringRE(string str1, str2, str3) Searches 'str1' and changes all instances matching 'str2' to the string 'str3'. Returns the number of changes made or -1 for error. The search is case sensitive. Regular expressions ARE recognized in 'str2'. If 'str3' is an empty string, the effect will be to delete all occurrences of 'str2' within 'str1'.
ChangeStringREi()	int ChangeStringREi(string str1, str2, str3) Searches 'str1' and changes all instances matching 'str2' to the string 'str3'. Returns the number of changes made or -1 for error. The search is case insensitive. Regular expressions ARE recognized in 'str2'. If 'str3' is an empty string, the effect will be to delete all occurrences of 'str2' within 'str1'.
CheckWord	Issues the Check Word command
ClearAllBookmarks	Issues the Clear All Bookmarks command
ClearAllClipboards	Issues the Clear All Clipboards command
ClearClipboard()	ClearClipboard(int n) Clears the content of Clipboard 'n'. The Windows Clipboard is number 0; private clipboards are numbered 1 to 8.
ClearClosedTabsList	Issues the Clear Closed Tabs List
ClearRecentFilesList	Issues the Clear Recent Files List command
ClearRecentProjectsList	Issues the Clear Recent Projects List command
ClearUndo	Issues the Clear Undo command
Close	Issues the Close command
CloseAll	Issues the Close All command
CloseAllButActive	Issues the Close All But Active command
ColorChart	Issues the HTML Color Chart command

Column	int Column Returns the column number of the text cursor in the current file, or -1 for error. The column returned is 1-based, not 0-based, and does not give consideration to the display value of any tabs that may appear in the line. See also DisplayColumn().
Comment	Issues the Comment command
ConfigureColors	Issues the Configure Colors command
ConfigureCtagsFunctionIndexing	Issues the Configure Ctags Function Indexing command
ConfigureKeyboard	Issues the Configure Keyboard command
ConfigurePreferences	Issues the Configure Preferences command
ConfigurePrinterFont	Issues the Configure Printer Font command
ConfigureScreenFont	Issues the Configure Screen Font command
ConfigureSyntaxHighlighting	Issues the Configure Syntax Highlighting command
ConfigureTemplates	Issues the Configure Templates command
ConfigureTextHighlighting	Issues the Configure Text Highlighting command
ConfigureToolbar	Issues the Configure Toolbar command
ConfigureUserTools	Issues the Configure User Tools command
Copy	Copy Copy the selected text to the current clipboard. If text is not selected, the current line is copied to the clipboard.
CopyFile()	int CopyFile(string oldname, string newname) Copies the file 'oldname' to the file 'newname', overwriting the output file if it already exists. Returns 1 for success or -1 for error.
CopyFilename	Issues the Copy Filename command
cos()	float cos(float x) Returns the cosine of 'x'. The angle 'x' must be in radians.
cosh()	float cosh(float x) Returns the hyperbolic cosine of 'x'. The angle 'x' must be in radians.
CreateDirectory()	int CreateDirectory(string dir) Creates a new directory according to the fully qualified filepath in 'dir'. Returns 1 for success or -1 for error.

CtagsFunctionIndex	Issues the Ctags Function Index command
Cut	Cut Cut the selected text to the current clipboard. If text is not selected, the current line is cut to the clipboard.
CutAppend	CutAppend Cut the selected text and append it to the current clipboard. If text is not selected, the current line is cut and appended to the clipboard.
Declaration	Issues the Declaration command
Decrement()	int Decrement(int n) Subtracts 'n' from the value at the text cursor and places the result in the text file. Returns the result of the operation or -1 for error. If 'n' is not supplied the Decrement dialog will appear when the macro is run.
Delete	Delete Deletes the character at the text cursor, or the selected text.
DeleteBlankLines	Issues the Delete Blank Lines command
DeleteBookmarkedLines	Issues the Delete Bookmarked Lines command
DeleteDuplicateLines	Issues the Delete Duplicate Lines command
DeleteFile()	int DeleteFile(string name) Deletes the fully qualified filepath 'name' from the disk, without requesting confirmation. Returns 1 for success or -1 for error.
DeleteLine	int DeleteLine(int n) Deletes line 'n' in the current file. Returns 1 for success or -1 for error. If 'n' is not supplied, the current line is deleted.
DeleteLinesThatBeginWith	int DeleteLinesThatBeginWith(string str) Delete lines that begin with the string 'str'. Returns 1 for success or -1 for error. If 'str' is not supplied a dialog will appear when the macro is run.
DeleteLinesThatContain	int DeleteLinesThatContain(string str) Delete lines that contain the string 'str'. Returns 1 for success or -1 for error. If 'str' is not supplied a dialog will appear when the macro is run.
DeleteLinesThatDoNotBeginWith	int DeleteLinesThatDoNotBeginWith(string str) Delete lines that do not begin with the string 'str'. Returns 1 for success or -1 for error. If 'str' is not supplied a dialog will appear when the macro is run.
DeleteLinesThatDoNotContain	int DeleteLinesThatDoNotContain(string str) Delete lines that do not contain the string 'str'.

	Returns 1 for success or -1 for error. If 'str' is not supplied a dialog will appear when the macro is run.
DeleteLinesThatDoNotEndWith	int DeleteLinesThatDoNotEndWith(string str) Delete lines that do not end with the string 'str'. Returns 1 for success or -1 for error. If 'str' is not supplied a dialog will appear when the macro is run.
DeleteLinesThatEndWith	int DeleteLinesThatEndWith(string str) Delete lines that end with the string 'str'. Returns 1 for success or -1 for error. If 'str' is not supplied a dialog will appear when the macro is run.
DeleteNextWord	Issues the Delete Next Word command
DeletePreviousWord	Issues the Delete Previous Word command
DeleteToEndOfLine	Issues the Delete to End of Line command
DeleteToStartOfLine	Issues the Delete to Start of Line command
Deselect()	Deselect(int mode) Releases the text selection, if one exists. If 'mode' is 0, the text cursor is placed at the beginning of the selection. If 'mode' is 1, the text cursor is placed at the end of the selection. If 'mode' is 2, the current position of the text cursor is maintained. Returns 1 for success or -1 for error.
DisplayColumn	int DisplayColumn Returns the column number of the text cursor in the current file, or -1 for error. The column returned is 1-based, not 0-based, and gives consideration to the display value of any tabs that may appear in the line. See also Column().
Divide()	int Multiply(int n) Divides the value at the text cursor by 'n' and places the result in the text file. Returns the result of the operation or -1 for error. If 'n' is not supplied the Divide dialog will appear when the macro is run.
Down	int Down(int n) Issues the Down command 'n' times. Returns the number of commands performed or -1 for error. The argument 'n' is optional; if it is not provided a single command is performed.
DuplicateAndIncrement	Issues the Duplicate and Increment command
DuplicateLine	Issues the Duplicate Line command
e	float e Returns the value of Euler's number 'e', which is approximately 2.7182818285.

EBCDICtoASCII	Issues the EBCDIC to ASCII command
EditClipboard()	EditClipboard(int n) Opens Clipboard 'n' in a window for editing. The Windows Clipboard is number 0; private clipboards are numbered 1 to 8.
EndOfFile	Issues the End of File command
EndOfLine	Issues the End of Line command
Enter	Issues the Enter command
EraseValue()	int EraseValue(string name) Erases the variable 'name' from the macro variable storage area. Returns 1 for success or -1 for error. See also ReadValue(), WriteValue(), ValueExists().
ErrorChart	Issues the Error Chart command
Exit	Exit Issues the File|Exit command to close the editor. If one or more files have not been saved a prompt will appear when the macro is run.
exp()	float exp(float x) Returns the value 'e' raised to the 'x'.
ExploreDataFolder	Issues the Explore Data Folder command
ExploreProgramFolder	Issues the Explore Program Folder command
ExtractDrive()	int ExtractDrive(string str) Converts the string 'str' so that it contains only the drive designation portion of itself (eg 'C:'). Returns the length of 'str' or -1 for error.
ExtractFileExt()	int ExtractFileExt(string str) Converts the string 'str' so that it contains only the file extension portion of itself. The leading '.' is retained in the resulting string. Returns the length of 'str' or -1 for error.
ExtractFileNameAndExt()	int ExtractFileNameAndExt(string str) Converts the string 'str' so that it contains the filename.ext portion of itself. Returns the length of 'str' or -1 for error.
ExtractFileNameOnly()	int ExtractFileNameOnly(string str) Converts the string 'str' so that it contains only the filename portion of itself. Returns the length of 'str' or -1 for error.
ExtractFilePath()	int ExtractFilePath(string str) Converts the string 'str' so that it contains only the filepath portion of itself. The trailing backslash is retained in the resulting string. Returns the length

	of 'str' or -1 for error.
fabs()	float fabs(float x) Returns the absolute value of 'x'.
factorial()	int factorial(int x) Returns the value of x factorial, also known as x! Returns -1 for error or overflow.
FastFrame()	int FastFrame(int style) Surrounds the columnar selection with a frame according to 'style'. When 'style' is in the range 1 to 11, a corresponding line style from the Fast Frame dialog is used. If 'style' is not supplied the Fast Frame dialog will appear when the macro runs. Returns 1 for success or -1 for error.
FileCount	int FileCount Returns the number of files currently open in the editor.
FileExists()	int FileExists(string filepath) Returns 1 if 'filepath' exists, 0 if it does not exists, or -1 for error.
FileName	int FileName(string fn) Fills 'fn' with the full path of the current file. Returns the length of the filepath or -1 for error.
FilePicker	Issues the File Picker command
FileProperties	Issues the File Properties command
FileTabsBottom	Issues the File Tabs Bottom command
FileTabsTop	Issues the File Tabs Top command
FillWithString()	int FillWithString(string str) Fills the selected region with 'str'. Returns 1 for success or -1 for error. If 'str' is not supplied the Fill with String dialog will appear when the macro is run.
Find()	int Find(string str) Searches for string 'str'. Returns TRUE if found, FALSE if not found, -1 for error. If 'str' is not supplied the Find dialog will appear when the macro is run. Find() will use the current settings on the Find dialog, but it will force the Perl Regular Expressions option OFF, and the Match Case option ON. See also Findi(), FindRE() and FindREi().
FindADiskFile	Issues the Find a Disk File command
FindAndCount()	int FindAndCount(string str) Searches for occurrences of 'str' and returns the

	number found. If 'str' is not supplied the Find and Count dialog will appear when the macro is run.
FindDifferingLines	int FindDifferingLines() Issues the Find Differing Lines command and returns 1 if a mismatch is found, or 0 if no additional mismatches exist.
FindDistinctLines	Issues the Find Distinct Lines command
FindDuplicateLines	Issues the Find Duplicate Lines command
FindFast	Issues the Find Fast command
Findi()	int Findi(string str) Searches for string 'str'. Returns TRUE if found, FALSE if not found, -1 for error. If 'str' is not supplied the Find dialog will appear when the macro is run. Findi() will use the current settings on the Find dialog, but it will force the Perl Regular Expressions option OFF, and the Match Case option OFF. See also Find(), FindRE() and FindREi().
FindMate	int FindMate search for a mate to the parenthetical sequence at the text cursor. Returns TRUE if found, FALSE if not found.
FindNext	int FindNext Returns 1 if the string is found, 0 if not found, -1 for error.
FindPrevious	int FindPrevious Returns 1 if the string is found, 0 if not found, -1 for error.
FindRE()	int FindRE(string str) Searches for string 'str'. Returns TRUE if found, FALSE if not found, -1 for error. If 'str' is not supplied the Find dialog will appear when the macro is run. FindRE() will use the current settings on the Find dialog, but it will force the Perl Regular Expressions option ON, and the Match Case option ON. See also Find(), Findi() and FindREi().
FindREi()	int FindREi(string str) Searches for string 'str'. Returns TRUE if found, FALSE if not found, -1 for error. If 'str' is not supplied the Find dialog will appear when the macro is run. FindREi() will use the current settings on the Find dialog, but it will force the Perl Regular Expressions option ON, and the Match Case option

	OFF. See also Find(), Findi() and FindRE().
FindTextInDiskFiles	Issues the Find Text in Disk Files command
FindUniqueLines	Issues the Find Unique Lines command
FlipCase	Issues the Flip Case command
floor()	float floor(float x) Returns (as a float) the largest integer not greater than 'x'. Example: floor(1.5) returns 1.0.
FormatXML	Issues the Format XML command
Formfeed	Issues the Formfeed command
FTPOpen()	int FTPOpen(string fn) Opens the FTP file 'fn' for editing. If 'fn' is already open for editing, its window will become the current window. Returns 1 for success or -1 for error. Remember: \\ must be used to denote \ within 'fn'.
GetChar()	int GetChar(string prompt, char c) Displays a message box with 'prompt' and fills 'c' with the character entered by the user. Returns 1 for success or -1 for error. See also PressChar().
GetClipboardText()	int GetClipboardText(string str, int n) Fills 'str' with the text of Clipboard 'n'. Returns the length of the text installed or -1 for error. The Windows Clipboard is number 0; private clipboards are numbered 1 to 8.
GetCurrentDirectory()	int GetCurrentDirectory(string str) Retrieves the current directory for the active process and places it in 'str'. Returns 1 for success or -1 for error. See also SetCurrentDirectory().
GetDataDirectory()	int GetDataDirectory(string str) Fills 'str' with the full path of the data directory. Returns 1 for success or -1 for error. See also GetProgramDirectory().
GetDate()	int GetDate(int y, int m, int d) Gets the current date and fills 'y', 'm' and 'd' with the year, month and date, respectively. Returns 1 for success or -1 for error.
GetDayName()	int GetDayName(string str, int n) Fills 'str' with the 3-character name of weekday

	number 'n' (1-7). The string returned is sensitive to the local language. Returns 1 for success or -1 for error.
GetEditMode()	int GetEditMode() Returns the edit mode of the current file. Returns 0 if the edit mode is Insert, or 1 if the edit mode is Typeover. Returns -1 if a file is not open.
GetEnv()	int GetEnv(string str1, string str2) Fills 'str1' with the content of the environment variable named in 'str2'. Returns 1 for success or -1 for error.
GetFloat()	int GetFloat(string prompt, float x) Displays a message box with 'prompt' and fills 'x' with the value entered by the user. Returns 1 for success or -1 for error.
GetGMTDateTime()	int GetGMTDateTime(int y, int m, int d, int hh, int mm, int ss) Gets the current date and time at GMT (Greenwich Mean Time) and fills 'y', 'm', 'd', 'hh', 'mm' and 'ss' with year/month/day/hour/minute/second, respectively. Hours will be in 24-hour format. Returns 1 for success or -1 for error.
GetInt()	int GetInt(string prompt, int n) Displays a message box with 'prompt' and fills 'n' with the value entered by the user. Returns 1 for success or -1 for error.
GetLineText()	int GetLineText(int n, string str) Fills 'str' with the text of line 'n'. Returns the length of line 'n' or -1 for error.
GetMonName()	int GetMonName(string str, int n) Fills 'str' with the 3-character name of month number 'n' (1-12). The string returned is sensitive to the local language. Returns 1 for success or -1 for error.
GetMonthName()	int GetMonthName(string str, int n) Fills 'str' with the full name of month number 'n' (1-12). The string returned is sensitive to the local language. Returns 1 for success or -1 for error.
GetProgramDirectory()	int GetProgramDirectory(string str) Fills 'str' with the full path of the program directory. Returns 1 for success or -1 for error. See also GetDataDirectory().
GetReadOnly	int GetReadOnly Returns the read-only state of the current file. Returns 1 if the file is read-only, 0 if the file is not

	read-only, or -1 for error. See also: ToggleReadOnly()
GetSelection()	int GetSelection(string str) Fills 'str' with the currently selected text. Returns the length of the selection or -1 for error. See also: ActiveClipboard
GetSelectionBounds()	int GetSelectionBounds(int l1, int c1, int l2, int c2) Fills l1, c1, l2, c2 with the bounds of the currently selected text. Returns 1 for success or -1 for error.
GetSelectionMode()	int GetSelectionMode() Returns 0 if the current selection mode is Stream, 1 if the current selection mode is Columnar
GetSelectionSize	int GetSelectionSize Returns the number of character currently selected, 0 if a selection is not present, or -1 for error.
GetShortName()	int GetShortName(string shortname, string fullname) Fills 'shortname' with the 8.3/DOS format short filename that corresponds to 'longname'. Returns 1 for success, or -1 for error.
GetString()	int GetString(string prompt, string result [, string default]) Displays a message box with 'prompt' and fills 'result' with the string entered by the user. If the optional third parameter 'default' is present, it is suggested as the default entry string. Returns the length of 'result' or -1 for error.
GetTextWidth	int GetTextWidth Returns the current Text Width value.
GetTime12()	int GetTime12(int h, int m, int s, int pm) Gets the current time and fills 'h', 'm' and 's' with hours, minutes and seconds, respectively. Hours will be in 12-hour format. If the time is PM, 'pm' is set to 1, else it is set to 0. Returns 1 for success or -1 for error.
GetTime24()	int GetTime24(int h, int m, int s) Gets the current time and fills 'h', 'm' and 's' with hours, minutes and seconds, respectively. Hours will be in 24-hour format. Returns 1 for success or -1 for error.
GetWeekday()	int GetWeekday(int y, int m, int d) Returns the number of the weekday associated with the date 'y', 'm', 'd'. Returns 1-7 for success or -1 for error.

GetWeekdayName()	int GetWeekdayName(string str, int n) Fills 'str' with the full name of weekday number 'n' (1-7). The string returned is sensitive to the local language. Returns 1 for success or -1 for error.
GetWindowNumber()	int GetWindowNumber(string fn) Returns the window number that holds the file 'fn'. Returns -1 for error, 0 if the named file is not open, or a postive value if the file's window is located. See also Filename(), SwitchToWindow().
GetWord()	int GetWord(string str) Fills 'str' with the word at the text cursor. Returns the length of the word found or -1 for error. See also SelectWord().
GetWordDelimiters()	int GetWordDelimiters(string str) Fills 'str' with a string that contains the characters considered to be word delimiters for the current file. Returns 1 for success or -1 for error.
GetYesNo()	int GetYesNo(string title, string query) Gets a Yes or No reply from the user. Displays a message box with title 'title' and message 'query'. Returns 1 if the user clicks Yes, 0 if the user clicks No.
GoToByteOffset()	int GoToByteOffset(int n OR string str) Go to offset 'n' in the current file. Returns 1 for success or -1 for error. A string parameter is also accepted. For example: "+25" will cause the cursor to be moved ahead 25 bytes. If a parameter is not provided the Go to Byte Offset dialog will appear when the macro is run.
GoToColumn()	int GoToColumn(int n OR string str) Go to column 'n' in the current file. Returns 1 for success or -1 for error. A string parameter is also accepted. For example: "-12" will cause the cursor to be moved 12 columns to the left. If a parameter is not provided the Go to Column dialog will appear when the macro is run.
GoToLine()	int GoToLine(int n or string str) Go to line 'n' in the current file. Returns 1 for success or -1 for error. A string parameter is also accepted. For example: "+50" will cause the cursor to be moved ahead 50 lines. If a parameter is not provided the Go to Line dialog will appear when the macro is run.
GoToParagraph()	int GoToParagraph(int n) Go to paragraph 'n' in the current file. Returns 1 for

	success or -1 for error. If a parameter is not provided the Go to Paragraph dialog will appear when the macro is run.
HardenLineEnders	Issues the Harden Line Enders command.
HTMLImageTag()	int HTMLImageTag(string fn) Inserts an HTML 'IMG' tag for the image file 'fn'. BMP, GIF and JPG images are supported. Returns 1 for success or -1 for error.
Increment()	int Increment(int n) Adds 'n' to the value at the text cursor and places the result in the text file. Returns the result of the operation or -1 for error. If 'n' is not supplied the Increment dialog will appear when the macro is run.
IndentOneSpace	Issues the Indent One Space command
IndentOneTabstop	Issues the Indent One Tabstop command
IndentWithString()	int IndentWithString(string str) Indents the selected lines with 'str'. Returns 1 for success or -1 for error. If 'str' is not supplied the Indent with String dialog will appear when the macro is run.
InsertCharacter()	int InsertCharacter(char ch) Inserts character 'ch' into the edited text. Returns the ASCII value of 'ch' or -1 for error. (This command is identical to PutChar.)
InsertFile()	int InsertFile(string str) Insert file 'str' into the current file. Returns 1 for success or -1 for error. If 'str' is not provided the Insert File dialog will appear when the macro is run.
InsertFilename	Issues the Insert Filename command
InsertLineAbove	Issues the Insert Line Above command
InsertLineBelow	Issues the Insert Line Below command
InsertLongDate	Issues the Insert Long Date command
InsertLongTime	Issues the Insert Long Time command
InsertMode	InsertMode Switches the edit mode to Insert in the current file. See also ToggleEditMode().
InsertShortDate	Issues the Insert Short Date command
InsertShortTime	Issues the Insert Short Time command
InvertLines	Issues the Invert Lines command
isalnum()	int isalnum(char c)

	Returns non-zero if character 'c' is alphanumeric.
isalpha()	int isalpha(char c) Returns non-zero if character 'c' is alphabetic.
isascii()	int isascii(char c) Returns non-zero if character 'c' is in the range 0-127.
IsBookmarked()	int IsBookmarked(int n) Returns 1 if line 'n' is bookmarked, 0 if not, or -1 for error. If 'n' is not supplied, the current line is assumed.
iscntrl()	int iscntrl(char c) Returns non-zero if character 'c' is a control character (0-31 or 127).
isdigit()	int isdigit(char c) Returns non-zero if character 'c' is a digit.
islower()	int islower(char c) Returns non-zero if character 'c' is lowercase.
ispunct()	int ispunct(char c) Returns non-zero if character 'c' is punctuation.
isspace()	int isspace(char c) Returns non-zero if character 'c' is whitespace (space, tab, newline, etc.).
isupper()	int isupper(char c) Returns non-zero if character 'c' is uppercase.
isxdigit()	int isxdigit(char c) Returns non-zero if character 'c' is a hex digit (A-F, a-f, 0-9).
JustificationStyle()	int JustificationStyle(int n) Sets the current text justification style according to 'n': 1=Left, 2=Center, 3=Right, 4=Smooth. If 'n' is not supplied the Justification Style dialog will appear when the macro runs. Returns 1 for success or -1 for error.
LastCharacter()	int LastCharacter(string str) Returns the last character in 'str' or 0 if 'str' is an empty string.
Left	int Left(int n) Issues the Left command 'n' times. Returns the number of commands performed or -1 for error. The argument 'n' is optional; if it is not provided a single command is performed.
LeftWindowEdge	Issues the Left Window Edge command
LineContains()	int LineContains(int n, string str)

	Returns the number of occurrences of 'str' that appear in line 'n'. The search performed is case sensitive. Regular expressions are not recognized.
LineContainsi()	int LineContainsi(int n, string str) Returns the number of occurrences of 'str' that appear in line 'n'. The search performed is case insensitive. Regular expressions are not recognized.
LineContainsRE()	int LineContainsRE(int n, string str) Returns the number of matches to 'str' that appear in line 'n'. The search performed is case sensitive. Regular expressions ARE recognized.
LineContainsREi()	int LineContainsREi(int n, string str) Returns the number of matches to 'str' that appear in line 'n'. The search performed is case insensitive. Regular expressions ARE recognized.
LineCount	int LineCount Returns the number of lines in the current file.
LineDrawing()	int LineDrawing(int style) Initiates or terminates Line Drawing mode. If 'style' is 1 to 11, a corresponding line style from the Line Drawing dialog is activated. The Up, Down, Left and Right commands can then be used to draw lines and boxes. When 'style' is 0, Line Drawing mode is terminated. If 'style' is not supplied the Line Drawing dialog will appear when the macro runs. Returns 1 for success or -1 for error.
LineIsEmpty()	int LineIsEmpty(int n) Returns TRUE if line 'n' is empty. Note: a line containing only whitespace is considered empty.
LineLength()	int LineLength(int n) Returns the number of characters in line 'n'.
LineNumber	int LineNumber Returns the current line number in the current file or -1 for error.
LineSpacing	int LineSpacing(int mode) Formats the range of selected lines, or the whole file, according to 'mode'. 'mode' can be 1, 2 or 3, which produces single, double or triple spacing, respectively. Returns 1 for success or -1 for error.
log()	float log(float x) Returns the natural log of 'x'. 'x' must be a positive value greater than 0.
log10()	float log(float x) Returns the base 10 log of 'x'. 'x' must be a positive value greater than 0.

MakeLineBottom	Issues the Make Line Bottom command
MakeLineCenter	Issues the Make Line Center command
MakeLineTop	Issues the Make Line Top command
max()	int max(int n1, int n2) Returns the greater of 'n1' and 'n2'.
Maximize	Maximize the current editing window.
MaximizeAll	Issues the Maximize All command
Message()	Message(string str, ...) Displays a pop-up message box with title 'str' and a message that is built from all arguments that follow. Example: Message("Results", n, " removed;", m, " remain.");
min()	int min(int n1, int n2) Returns the lesser of 'n1' and 'n2'.
Minimize	Minimize the current editing window.
MinimizeAll	Issues the Minimize All command
Modified	int Modified Returns 1 if the current file has been modified, else 0. Returns -1 for error. See also SetModified().
MoveLineDown	Issues the Move Line Down command
MoveLineUp	Issues the Move Line Up command
Multiply()	int Multiply(int n) Multiplies the value at the text cursor by 'n' and places the result in the text file. Returns the result of the operation or -1 for error. If 'n' is not supplied the Multiply dialog will appear when the macro is run.
New	Issues the New command
NextBookmark	Issues the Next Bookmark command
NextFunction	Issues the Next Function command
NextParagraph	Issues the Next Paragraph command
OEMChart	Issues the OEM Chart command
OEMtoANSI	Issues the OEM to ANSI command
Open()	int Open(string fn) Opens the file 'fn' for editing. If 'fn' is already open for editing, its window will become the current window. Returns 1 for success or -1 for error. Remember: \\ must be used to denote \ within 'fn'.

OpenEmail()	int OpenEmail(string str) Initiates an email message to the address in 'str' using the default email client. Returns 1 for success or -1 for error. See also OpenEmailAtCursor.
OpenEmailAtCursor	Issues the Open Email at Cursor command
OpenFileInBrowser	Issues the Open File in Browser command
OpenFilenameAtCursor	Issues the Open Filename at Cursor command
OpenHeaderFile	Issues the Open Header File command
OpenHex()	int OpenHex(string fn) Opens the file 'fn' for hex mode viewing and editing. Returns 1 for success or -1 for error. Remember: \\ must be used to denote \ within 'fn'.
OpenProgramAtCursor	Issues the Open Program at Cursor command
OpenRecentFile()	OpenRecentFile(int n) Opens recent file number 'n'. When a sufficient file history exists, 'n' can range from 1 to 24.
OpenRecentProject()	OpenRecentProject(int n) Opens recent project number 'n'. When a sufficient project history exists, 'n' can range from 1 to 16.
OpenSystemFiles	Issues the Open System Files command
OpenURL()	int OpenURL(string str) Opens the URL described in 'str' in the default internet browser. Returns 1 for succes or -1 for error. See also OpenURLAtCursor.
OpenURLAtCursor	Issues the Open URL at Cursor command
PageDown	Issues the Page Down command
PageLeft	Issues the Page Left command
PageRight	Issues the Page Right command
PageSetup	Issues the Page Setup command
PageUp	Issues the Page Up command
Paste	Issues the Paste command
PasteAs	Issues the Paste As command
PasteClipboard()	PasteClipboard(int n) Pastes the content of Clipboard 'n' into the current file. The Windows Clipboard is number 0; private clipboards are numbered 1 to 8.

Pause	Pause Pauses macro execution by displaying a message box and waiting for it to be closed.
pi	float pi Returns the value of pi, which is approximately 3.1415926536.
PlaySound()	int PlaySound(string filepath) Plays the .WAV file described in 'filepath'. Returns 1 for success or -1 for error.
pow()	float pow(float x, float y) Returns the value of 'x' raised to the power 'y'.
PressChar()	int PressChar(string prompt, char c) Displays the message 'prompt' on the status bar and fills 'c' with the next character pressed by the user. A popup dialog does NOT appear. A file must be open in order for PressChar to operate. Returns 1 for success or -1 for error. Note: PressChar will not wait for a character when run in Debug mode. See also GetChar().
PreviousBookmark	Issues the Previous Bookmark command
PreviousFunction	Issues the Previous Function command
PreviousParagraph	Issues the Previous Paragraph command
Print	Issues the Print command
PrintAll	Issues the Print All command
PrintAllColor	Issues the Print All Color command
PrintAllMonochrome	Issues the Print All Monochrome command
PrintColor	Issues the Print Color command
printf()	int printf(string format, ...) Processes 'format' and inserts a string into the edited text, in accordance with the formatting commands used in 'C'. Returns the number of characters inserted. See the online help for more information.
PrintMonochrome	Issues the Print Monochrome command
PrintPreview	Issues the Print Preview command
PrintPreviewColor	Issues the Print Preview Color command
PrintPreviewMonochrome	Issues the Print Preview Monochrome command
PrintSetup	Issues the Print Setup command

ProjectAddAll	Issues the Project Add All command
ProjectAddOne	Issues the Project Add One command
ProjectAutoUpdate()	int ProjectAutoUpdate(int mode) Toggles the Auto-Update feature on or off for the active project according to 'mode'. Returns 1 for success or -1 for error.
ProjectClose	Issues the Project Close command
ProjectDelete()	int ProjectDelete(string name) Deletes the project file described by 'name'. A confirmation prompt will be presented. Returns 1 for success or -1 for error.
ProjectEditActive	Issues the Project Edit Active command
ProjectEditOther()	int ProjectEditOther(string name) Opens the project file described by 'name' for editing. If 'name' does not exist an empty file will be opened. Returns 1 for success or -1 for error.
ProjectName()	int ProjectName(string fn) Fills 'fn' with the full path of the active project file. Returns the length of the filepath or -1 for error.
ProjectNew	Issues the Project New command
ProjectOpen()	int ProjectOpen(string name) Open the project file described by 'name'. Returns 1 for success or -1 for error.
ProjectRemove	Issues the Project Remove command
ProjectUpdateAll	Issues the Project Update All command
ProjectUpdateOne	Issues the Project Update One command
PutChar()	int PutChar(char ch) Inserts character 'ch' into the edited text. Returns 1 for success or -1 for error.
PutClipboardText()	int PutClipboardText(string str, int n) Fills Clipboard 'n' with string 'str'. Returns the length of the text installed or -1 for error. The Windows Clipboard is number 0; private clipboards are numbered 1 to 8. See also AppendToClipboard().
PutFloat()	int PutFloat(float x) Inserts the value of 'x' into the edited text. Two decimal places will be used. Returns 1 for success or -1 for error. Use printf() if special formatting is required.
PutInt()	int PutInt(int n)

	Inserts the value of 'n' into the edited text. Returns 1 for success or -1 for error.
PutLineText()	int PutLineText(int n, string str) Replaces the text of line 'n' with 'str'. Returns the length of 'str' or -1 for error.
PutMany()	int PutMany(...) Inserts the supplied argument(s) into the edited text. Example: PutMany(5, " is a number", '\n');
PutSelection()	int PutSelection(string str) Inserts string 'str' into the edited text. Returns the length of 'str' or -1 for error. PutSelection() is the complement to GetSelection(), and it should be used instead of PutString to ensure proper insertion of column-selected text.
PutString()	int PutString(string str) Inserts string 'str' into the edited text. Returns the length of 'str' or -1 for error.
PutWordDelimiters()	int PutWordDelimiters(string str) Sets the word delimiters for the current file to the characters contained in 'str'. Returns 1 for success or -1 for error.
QuoteAndReformat	Issues the Quote and Reformat command
Random()	int Random(int n) Returns a random number between 0 and n-1 or -1 for error.
ReadValue()	int ReadValue(string name, char/int/string/float val) Reads a value from the macro variable storage area named 'name' and places it into variable 'val'. The type of 'val' must agree with the type used when the value was written using WriteValue(). Returns 1 for success or -1 for error. See also WriteValue(), EraseValue(), ValueExists().
Redo	Issues the Redo command
RedoAll	Issues the Redo All command
Reference	Issues the Reference command
Reformat	Issues the Reformat command
ReloadFile	Issues the Reload File command
RenameFile()	int RenameFile(string oldname, string newname) Renames the file or directory named 'oldname' to 'newname'. Files can be renamed across drives; directories must be on the same drive. The target 'newname' must not exist. Returns 1 for success or

	-1 for error.
Replace()	int Replace(string str1, string str2) Searches for 'str1' and replaces it with 'str2'. Returns the number of replacements made or -1 for error. The user will be prompted to confirm replacements. If 'str1' and 'str2' are not supplied the Replace dialog will appear when the macro is run. Replace() will use the current settings on the Replace dialog, but it will force the Perl Regular Expressions option OFF, and the Match Case option ON. See also Replacei().
ReplaceAgain	Issues the Replace Again command
ReplaceAll()	int ReplaceAll(string str1, string str2) Searches for 'str1' and replaces it with 'str2'. Returns the number of replacements made or -1 for error. The user will NOT be prompted to confirm replacements. ReplaceAll() will use the current settings on the Replace dialog, but it will force the Perl Regular Expressions option OFF, and the Match Case option ON. See also ReplaceAlli().
ReplaceAlli()	int ReplaceAlli(string str1, string str2) Searches for 'str1' and replaces it with 'str2'. Returns the number of replacements made or -1 for error. The user will NOT be prompted to confirm replacements. ReplaceAlli() will use the current settings on the Replace dialog, but it will force the Perl Regular Expressions option OFF, and the the Match Case option OFF. See also ReplaceAll().
ReplaceAllRE()	int ReplaceAllRE(string str1, string str2) Searches for 'str1' and replaces it with 'str2'. Returns the number of replacements made or -1 for error. The user will be prompted to confirm replacements. If 'str1' and 'str2' are not supplied the Replace dialog will appear when the macro is run. ReplaceAllRE() will use the current settings on the Replace dialog, but it will force the Perl Regular Expressions option to ON, and the Match Case option ON. See also ReplaceAll(), ReplaceAlli and ReplaceAllREi().
ReplaceAllREi()	int ReplaceAllREi(string str1, string str2)

	Searches for 'str1' and replaces it with 'str2'. Returns the number of replacements made or -1 for error. The user will be prompted to confirm replacements. If 'str1' and 'str2' are not supplied the Replace dialog will appear when the macro is run. ReplaceAllREi() will use the current settings on the Replace dialog, but it will force the Perl Regular Expressions option to ON, and the Match Case option OFF. See also ReplaceAll(), ReplaceAlli and ReplaceAllRE().
Replacei()	int Replacei(string str1, string str2) Searches for 'str1' and replaces it with 'str2'. Returns the number of replacements made or -1 for error. The user will be prompted to confirm replacements. If 'str1' and 'str2' are not supplied the Replace dialog will appear when the macro is run. Replacei() will use the current settings on the Replace dialog, but it will force the Perl Regular Expressions option OFF, and the Match Case option OFF. See also Replace().
ReplaceLineEnders()	int ReplaceLineEnders(string str1, string str2) Searches for 'str1' and replaces it with 'str2'. 'str1' and 'str2' may use the sequence \n (if within a quoted string) to represent a line ender. Returns the number of replacements made or -1 for error. The user will be NOT prompted to confirm replacements. If 'str1' and 'str2' are not supplied the Replace Line Enders dialog will appear when the macro is run. ReplaceLineEnders() will use the current settings on the Replace Line Enders dialog.
ReplaceRE()	int ReplaceRE(string str1, string str2) Searches for 'str1' and replaces it with 'str2'. Returns the number of replacements made or -1 for error. The user will be prompted to confirm replacements. If 'str1' and 'str2' are not supplied the Replace dialog will appear when the macro is run. ReplaceRE() will use the current settings on the Replace dialog, but it will force the Perl Regular Expressions option to ON, and the Match Case option ON. See also Replace(), Replacei() and ReplaceREi().
ReplaceREi()	int ReplaceREi(string str1, string str2) Searches for 'str1' and replaces it with 'str2'. Returns the number of replacements made or -1 for

	error. The user will be prompted to confirm replacements. If 'str1' and 'str2' are not supplied the Replace dialog will appear when the macro is run. ReplaceREi() will use the current settings on the Replace dialog, but it will force the Perl Regular Expressions option to ON, and the Match Case option OFF. See also Replace(), Replacei() and ReplaceRE().
Restore	Restore the current window from a minimized or maximized state.
RestoreAll	Issues the Restore All command
RestoreSettings	RestoreSettings Restores a variety of editor settings which were earlier noted using SaveSettings(). These functions can be used to ensure that a macro does not alter the editor's settings. The following settings are restored: Wordrwrap, Text Width, Justification Style, Edit Mode, Tab Display Size, Selection Mode, Active Clipboard and Word Delimiters.
Right	int Right(int n) Issues the Right command 'n' times. Returns the number of commands performed or -1 for error. The argument 'n' is optional; if it is not provided a single command is performed.
RightWindowEdge	Issues the Right Window Edge command
ROT5	Applies a ROT5 (rotation 5) conversion to the selected text
ROT13	Applies a ROT13 (rotation 13) conversion to the selected text
ROT18	Applies a ROT18 (rotation 18) conversion to the selected text
ROT47	Applies a ROT47 (rotation 47) conversion to the selected text
Round()	float Round(float x, int n) Returns the value of 'x' rounded to 'n' decimal places.
RunProgram()	int RunProgram(string fn, string params, string workdir, int wait) Runs the named program, document or folder in 'fn' by passing it to the ShellExecuteEx Windows API call. Command line parameters can be passed in 'params'. The program's working directory can be passed in 'workdir'. If 'wait' is 1, macro execution will be suspended until the program has been

	closed. Returns the completion code of ShellExecuteEx: non-zero for success, zero for error. See also OpenProgramAtCursor().
Save	Issues the Save command
SaveACopyAs()	int SaveACopyAs(string fn) Saves a copy of the current file to the file 'fn'. The name of the current file is not changed. Returns 1 for success or -1 for error. Remember: \\ must be used to denote \ within 'fn'.
SaveAll	Issues the Save All command
SaveAs()	int SaveAs(string fn) Saves the current file to the file 'fn'. The name of the current file is changed to 'fn'. Returns 1 for success or -1 for error. Remember: \\ must be used to denote \ within 'fn'.
SaveSelectionAs()	int SaveSelectionAs(string fn) Saves the current selection to the file 'fn'. Returns 1 for success or -1 for error. Remember: \\ must be used to denote \ within 'fn'.
SaveSettings	SaveSettings Records a variety of editor settings for later restoration using RestoreSettings(). These functions can be used to ensure that a macro does not alter the editor's settings. The following settings are saved: Wordrwrap, Text Width, Justification Style, Edit Mode, Tab Display Size, Selection Mode, Active Clipboard and Word Delimiters.
ScrollDown	Issues the Scroll Down command
ScrollLeft	Issues the Scroll Left command
ScrollRight	Issues the Scroll Right command
ScrollUp	Issues the Scroll Up command
SelectAllText	Issues the Select All Text command
SelectColumnar	Issues the Select Columnar command
SelectDown	Issues the Select Down command
SelectLeft	Issues the Select Left command
SelectPageDown	Issues the Select Page Down command
SelectPageLeft	Issues the Select Page Left command
SelectPageRight	Issues the Select Page Right command
SelectPageUp	Issues the Select Page Up command
SelectRight	Issues the Select Right command

SelectStream	Issues the Select Stream command
SelectToBottomOfPage	Issues the Select to Bottom of Page command
SelectToEndOfFile	Issues the Select to End of File command
SelectToEndOfLine	Issues the Select to End of Line command
SelectToStartOfFile	Issues the Select to Start of File command
SelectToStartOfLine	Issues the Select to Start of Line command
SelectToTopOfPage	Issues the Select to Top of Page command
SelectUp	Issues the Select Up command
SelectWithoutShift	Issues the Select without Shift command
SelectWord	int SelectWord(string str) Selects the word at the text cursor and places it in 'str'. Returns the length of the word selected or 0 if no word could be found to select. See also GetWord().
SelectWordLeft	Issues the Select Word Left command
SelectWordRight	Issues the Select Word Right command
SetClipboard()	SetClipboard(int n) Sets the active clipboard to Clipboard 'n'. The Windows Clipboard is number 0; private clipboards are numbered 1 to 8.
SetClipboardNext	Issues the Set Clipboard Next command
SetClipboardPrevious	Issues the Set Clipboard Previous command
SetCurrentDirectory	int SetCurrentDirectory(string str) Sets the current directory for the active process to 'str'. Returns 1 for success or -1 for error. See also GetCurrentDirectory().
SetModified	SetModified Sets the modifed state of the current to true. This might be used to force a Save operation even when a file has not been modified. Returns 1 for success or -1 for error. See also Modified().
ShadedTabZones()	int ShadedTabZones(int mode) Enables or disables the Shaded Tab Zones display mode according to 'mode'.
sin()	float sin(float x) Returns the sine of 'x'. The angle 'x' must be in radians.

sinh()	float sinh(float x) Returns the hyperbolic sine of 'x'. The angle 'x' must be in radians.
SoftenLineEnders	Issues the Soften Line Enders command.
SortFileTabsByExt()	int SortFileTabsByExt(int mode) Enables or disables the sorting of File Tabs by extension according to 'mode'. If 'mode' is 1 the feature is enabled. If 'mode' is 0 the feature is disabled.
SortFileTabsByName()	int SortFileTabsByName(int mode) Enables or disables the sorting of File Tabs by name according to 'mode'. If 'mode' is 1 the feature is enabled. If 'mode' is 0 the feature is disabled.
SortFileTabsByUse()	int SortFileTabsByUse(int mode) Enables or disables the sorting of File Tabs by name according to 'mode'. If 'mode' is 1 the feature is enabled. If 'mode' is 0 the feature is disabled.
SortLines	Issues the Sort Lines command
Space	Issues the Space command. If a range of lines is selected, the range will be indented.
SpacesToTabs	Issues the Spaces to Tabs command
SpellChecker	Issues the Spell Checker command
SplitHorizontal	Issues the Split Horizontal command
SplitVertical	Issues the Split Vertical command
sprintf()	int sprintf(string str1, string format, ...) Processes 'format' and builds an output string in 'str1', in accordance with the formatting commands used in 'C'. Returns the length of 'str1' or -1 for error. See the online help for more information.
sqrt()	float sqrt(float x) Returns the positive square root of 'x'.
StartOfFile	Issues the Start of File command
StartOfLine	Issues the Start of Line command
StatusMessage()	StatusMessage(...) Displays a message in the status bar that is built from all arguments that follow. Example: StatusMessage(n, " were removed;", m, " remain.");
strcat()	int strcat(string str1, string str2) Concatenates 'str2' to 'str1'. Returns the length of 'str1' or -1 for error.
strchr()	int strchr(string str, char c) Returns the offset at which the character 'c' appears

	in 'str' or -1 if 'c' does not appear. See also strrchr().
strcmp()	int strcmp(string str1, string str2) Compares 'str1' to 'str2' with case sensitivity. Returns 0 if the strings are equal. Returns < 0 if 'str1' is less than 'str2'. Returns > 0 if 'str1' is greater than 'str2'.
strcmpi()	int strcmpi(string str1, string str2) Compares 'str1' to 'str2' without case sensitivity. Returns 0 if the strings are equal. Returns < 0 if 'str1' is less than 'str2'. Returns > 0 if 'str1' is greater than 'str2'.
strcpy()	int strcpy(string str1, string str2) Copies 'str2' to 'str1'. Returns the length of 'str1' or -1 for error.
StripHTMLTags	Issues the Strip HTML/XML Tags command
StripLeadingSpaces	Issues the Strip Leading Spaces command
StripTrailingSpaces	Issues the Strip Trailing Spaces command
strlen()	int strlen(string str) Returns the length of 'str'.
strlwr()	int strlwr(string str) Converts the string 'str' to lowercase. Returns the length of 'str'.
strncat()	int strncat(string str1, string str2, int n) Concatenates up to 'n' characters from 'str2' to 'str1'. Returns the length of 'str1' or -1 for error. A NULL byte is added after the characters added.
strncmp()	int strncmp(string str1, string str2, int n) Compares up to 'n' characters in 'str1' to 'str2' with case sensitivity. Returns 0 if the strings are equal. Returns < 0 if 'str1' is less than 'str2'. Returns > 0 if 'str1' is greater than 'str2'.
strncmpi()	int strncmpi(string str1, string str2, int n) Compares up to 'n' characters in 'str1' to 'str2' without case sensitivity. Returns 0 if the strings are equal. Returns < 0 if 'str1' is less than 'str2'. Returns > 0 if 'str1' is greater than 'str2'.
strncpy()	int strncpy(string str1, string str2, int n) Copies up to 'n' characters from 'str2' to 'str1'. Returns the length of 'str1' or -1 for error. A NULL byte is added after the characters copied.
strrchr()	int strrchr(string str, char c) Returns the offset at which the character 'c' last

	appears in 'str' or -1 if 'c' does not appear. See also strchr().
strrev()	int strrev(string str) Reverses the string 'str' in place. Example: The string 'Boxer' would be converted to 'rexoB'.
strstr()	int strstr(string str1, string str2) Searches string 'str1' for the substring 'str2' with case sensitivity. Returns the offset at which 'str2' is found or -1 if not found.
strstri()	int strstri(string str1, string str2) Searches string 'str1' for the substring 'str2' without case sensitivity. Returns the offset at which 'str2' is found or -1 if not found.
strupr()	int strupr(string str) Converts the string 'str' to uppercase. Returns the length of 'str'.
SubString()	int SubString(string str1, string str2, int index, int len) Fills 'str1' with up to 'len' characters from 'str2', starting at offset 'index'. The first character in a string is referred to by offset 0. Returns the new length of 'str1' or -1 for error. If 'index' is negative, and 'len' is positive, 'str1' will be filled with 'len' characters starting 'index' characters in from the end of 'str2'. If 'len' is negative, the value of 'index' is ignored, and 'str1' is filled with the rightmost 'len' characters from 'str2'.
SwapLines	Issues the Swap Lines command
SwapWords	Issues the Swap Words command
SwitchToWindow()	SwitchToWindow(int n) Makes window 'n' active. Windows are numbered from 1 to the number of open files. See also FileCount. See also GetWindowNumber().
SyntaxHighlightAs	Issues the Syntax Highlight As command
Tab	Issues the Tab command. If a range of lines is selected, the range will be indented.
TabDisplaySize()	int TabDisplaySize(string str) Sets the Tab Display Size for the current file

	according to 'str'. If a single value appears in 'str', tabs are set to fixed width with the value in 'str'. If a series of comma-separated values are found in 'str', variable width tab stops will be used. Returns TRUE for success, -1 for error. If 'str' is not supplied the Tab Display Size dialog will appear when the macro is run.
TabsToSpaces	Issues the Tabs to Spaces command
tan()	float tan(float x) Returns the tangent of 'x'. The angle 'x' must be in radians.
tanh()	float tanh(float x) Returns the hyperbolic tangent of 'x'. The angle 'x' must be in radians.
Templates	Issues the Templates command
TextIsSelected	int TextIsSelected Returns 1 if a stream selection is present, 2 if a columnar selection is present, or 0 if no selection is present. Returns -1 for error.
TextWidth()	int TextWidth(int n) Sets the Text Width to 'n'. Returns TRUE for success or -1 for error. If 'n' is not supplied the Text Width dialog will appear when the macro is run.
TileAcross	Issues the Tile Across command
TileDown	Issues the Tile Down command
ToggleBookmark()	int ToggleBookmark(int n, int state) Sets bookmark 'n' according to 'state'. Returns TRUE for success, -1 for error. If 'state' is ON, bookmark 'n' is placed on the current line. If 'state' is OFF bookmark 'n' is cleared, where ever it is.
ToggleEditMode()	ToggleEditMode(int state) Toggles the edit mode according to 'state'. If 'state' is 1, Insert mode is used. If 'state' is 0, Typeover mode is used. See also the functions InsertMode() and TypeoverMode().
ToggleReadOnly()	ToggleReadOnly(int state) Toggle read-only mode according to 'state'. If 'state' is 1, read-only mode is set. If 'state' is 0, read-only mode is released.
tolower()	int tolower(char c) Returns the lowercase mate to character 'c'.
ToolbarBottom	Issues the Toolbar Bottom command

ToolbarLeft	Issues the Toolbar Left command
ToolbarRight	Issues the Toolbar Right command
ToolbarTop	Issues the Toolbar Top command
TopLine	int TopLine Returns the line number of the first line in the editor window. Returns -1 for error.
TopOfPage	Issues the Top of Page command
TotalAndAverage	Issues the Total and Average command
TouchFile()	int TouchFile(string name) Touch (ie, update the timestamp of) the file named 'name'. Returns 1 for success or -1 for error.
toupper()	int toupper(char c) Returns the uppercase mate to character 'c'.
Trim()	int Trim(string str) Removes leading and trailing blanks from 'str'. Returns the new length of 'str'.
TrimLeft()	int TrimLeft(string str) Removes leading blanks from 'str'. Returns the new length of 'str'.
TrimRight()	int TrimRight(string str) Removes trailing blanks from 'str'. Returns the new length of 'str'.
Trunc()	float Trunc(float x, int n) Returns the value of 'x' truncated to 'n' decimal places.
TypeoverMode	TypeoverMode Switches the edit mode to Typeover in the current file. See also ToggleEditMode().
Uncomment	Issues the Uncomment command.
Undo	Issues the Undo command.
UndoAll	Issues the Undo All command.
UndoAllClosedTabs	Issues the Undo All Closed Tabs command.
UndoClosedTab()	UndoClosedTab(int n) Reopens recently closed file tab number 'n'. When a sufficient closed tab list exists, 'n' can range from 1 to 10.
UndoCloseTab	Issues the Undo Close Tab command.
Unformat	Issues the Unformat command.

UnformatXML	Issues the Unformat XML command.
UnhighlightMatches	Issues the Unhighlight Matches command.
Unindent	Issues the Unindent command.
Up	int Up(int n) Issues the Up command 'n' times. Returns the number of commands performed or -1 for error. The argument 'n' is optional; if it is not provided a single command is performed.
UserList()	UserList(int n) Opens User List window 'n'.
UserTool()	UserTool(int n) Runs User Tool number 'n'.
UserToolWait()	UserTool(int n) Runs User Tool number 'n' and waits for it to complete execution.
ValueAtCursor	int ValueAtCursor Returns the ASCII value of the character at the cursor, or -1 if a character is not available at the cursor.
ValueExists()	int ValueExists(string name) Looks for the variable 'name' in the macro variable storage area. Returns 1 if it exists, 0 if it does not exists, or -1 for error. See also ReadValue(), WriteValue(), EraseValue().
ViewBookmarks()	int ViewBookmarks(int mode) Enables or disables the viewing of Bookmarks according to 'mode'.
ViewFileTabs()	int ViewFileTabs(int mode) Enables or disables the viewing of File Tabs according to 'mode'.
ViewHexMode()	int ViewHexMode(int mode) Enables or disables read-only hex mode viewing according to 'mode'.
ViewHexRuler()	int ViewHexRuler(int mode) Enables or disables the viewing of the Hex Ruler according to 'mode'.
ViewTextRuler()	int ViewTextRuler(int mode) Enables or disables the viewing of the Text Ruler according to 'mode'.
ViewHScrollBar()	int ViewHScrollBar(int mode) Enables or disables the viewing of Horizontal Scroll Bars according to 'mode'.

ViewLineNumbers()	int ViewLineNumbers(int mode) Enables or disables the viewing of Line Numbers according to 'mode'.
ViewRightMarginRule()	int ViewRightMarginRule(int mode) Enables or disables the viewing of the Right Margin Rule according to 'mode'.
ViewStatusBar()	int ViewStatusBar(int mode) Enables or disables the viewing of the Status Bar according to 'mode'.
ViewSyntaxHighlighting()	int ViewSyntaxHighlighting(int mode) Enables or disables the Syntax Highlighting feature according to 'mode'.
ViewTextHighlighting()	int ViewTextHighlighting(int mode) Enables or disables the Text Highlighting feature according to 'mode'.
ViewToolbar()	int ViewToolbar(int mode) Enables or disables the viewing of the Toolbar according to 'mode'.
ViewVisibleSpaces()	int ViewVisibleSpaces(int mode) Enables or disables the viewing of Visible Spaces according to 'mode'.
ViewVScrollBar()	int ViewVScrollBar(int mode) Enables or disables the viewing of Vertical Scroll Bars according to 'mode'.
VisualWrap()	int VisualWrap(int mode) Enables or disables Visual Wrap according to 'mode'.
VisualWrapOptions	Issues the Visual Wrap Options command.
Wait()	int Wait(int n) Delays macro execution for 'n' milliseconds. 1000 milliseconds equals 1 second. Returns 1 for success or -1 for error.
WindowHeight	int WindowHeight Returns the number of lines that can be displayed in the current window.
WindowLastVisited	Issues the Window Last Visited command
WindowList	Issues the Window List command
WindowNext	Issues the Window Next command
WindowPrevious	Issues the Window Previous command
WindowSkip	int WindowSkip(int mode) If 'mode' is 1, sets the skip status for the current window to on. If mode is 0, skip status is turned off. Returns 1 for success or -1 for error.

WindowWidth	int WindowWidth Returns the number of columns that can be displayed in the current window.
WordCount()	int WordCount(int lines, int words, int chars) Fills the supplied variables with the count of lines, words and characters, respectively. Returns 1 for success or -1 for error.
WordLeft	int WordLeft(int n) Issues the Word Left command 'n' times. Returns the number of commands performed or -1 for error. The argument 'n' is optional; if it is not provided a single command is performed.
WordRight	int WordRight(int n) Issues the Word Right command 'n' times. Returns the number of commands performed or -1 for error. The argument 'n' is optional; if it is not provided a single command is performed.
WordWrap()	int WordWrap(int mode) Enables or disables Typing Wrap according to 'mode'. Deprecated: see TypingWrap().
WriteValue()	int WriteValue(string name, char/int/string/float val) Writes 'val' to the macro variable storage area named 'name'. 'name' will be visible to other macros, so be careful to choose a unique identifier. Returns 1 for success or -1 for error. See also ReadValue(), EraseValue(), ValueExists().
xtoi()	int xtoi(string str) Returns the integer value of the hexadecimal number described by string 'str'. Returns -1 for error.

6.17 Macro Language Reference

Data Types

Boxer's Macro Language supports the following data types:

```
string
char
int
float
```

A `string` can hold a series of characters up to 2,048 bytes in length. The end of a string is marked with a Null character (ASCII 0). A string constant is enclosed within double quotes.

The `char` data type is an 8-bit, unsigned data type which can hold values in the range 0 to 255. A character constant is enclosed within single quotes.

The `int` data type is a 32-bit, signed data type which can hold integer values in the range -2,147,483,648 to 2,147,483,647.

The `float` data type is a double precision, signed data type that can hold values in the range 2.2250738585072014e-308 to 1.7976931348623158e+308.

Keywords

The following words are reserved keywords and may not be used as variable names:

```
break       int         true
continue    char        false
do          string      yes
else        float       no
for         void        on
goto                    off
if
macro
return
while
```

The keywords listed above are case sensitive, and must be entered in lowercase. The symbolic constants in the third column (`true, false, yes, no, on, off`) are an exception: they can appear in lowercase, uppercase, or even in mixed case.

Arithmetic Operators

The following arithmetic operators are supported:

Operator	Meaning
+	addition
-	subtraction
*	multiplication
/	division
%	modulus
++	increment
--	decrement

The modulus operator (`%`) returns the remainder from an integer division operation. For example, the expression `n = 7 % 4` will result in `n` receiving the value `3`, since `7 / 4` leaves a remainder of `3`.

The increment and decrement operators can be used to increase or decrease an integer

variable by 1. The expression:

```
i++;
```

is equivalent to:

```
i = i + 1;
```

The ++ and -- operators can be used in either prefix or postfix location. If i has an initial value of 3, the statement:

```
n = i++;
```

will leave n with the value of 3, while i is incremented to 4. The incrementing of i occurs *after* the assignment due to the postfix location.

Assuming i again starts with a value of 3, the statement:

```
n = --i;
```

will leave n with a value of 2 and i with a value of 2. The decrementing of i occurs *before* the assignment due to the prefix location.

The addition (+) operator has been overloaded to support string concatenation. The following statements:

```
string s1 = "Boxer ";
string s2 = "Text Editor";
string s3 = s1 + s2;
```

would result in s3 having the value: "Boxer Text Editor"

Assignment Operators

The following assignment operators are supported:

Operator	Meaning
=	assignment
+=	addition assignment
-=	subtraction assignment
*=	multiplication assignment
/=	division assignment
%=	modulus assignment
&=	bitwise AND assignment
|=	bitwise OR assignment

^=	bitwise XOR assignment
<<=	left shift assignment
>>=	right shift assignment

The assignment operator (=) should be familiar to all. The other operators which each conclude with = all represent a shorthand notation. For example, the statement:

```
i += 5;
```

is equivalent to:

```
i = i + 5;
```

The += operator has been *overloaded* to support string concatenation. The following statements:

```
string str = "Boxer ";
str += "Text Editor";
```

would result in `str` having the value: `"Boxer Text Editor"`

The last five operators listed above are bitwise assignment operators. Their function is analogous to the += operator; see the Bitwise Operators section of this topic for some additional detail.

Boolean Operators

The following Boolean operators are supported:

Operator	Meaning
==	equal
!=	not equal
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
&&	logical AND
||	logical OR
!	logical NOT (unary negation)
~=	case insensitive string comparison

The operators ==, !=, <, >, <= and >= have been overloaded to allow operations on strings. A string is considered greater than another string if it would appear higher

in an alphabetic sort. In other words, the statement:

```
if ("apple" < "zebra")
```

evaluates to TRUE.

The first nine operators above are standard to most high-level languages. The last operator is specific to Boxer's Macro Language, and permits strings to be compared without case sensitivity. For example, the statement:

```
if ("MasterCard" ~= "mastercard")
```

would evaluate to TRUE.

Bitwise Operators

The following bitwise operators are supported:

Operator	Meaning
&	bitwise AND
|	bitwise OR
^	bitwise XOR
<<	left shift
>>	right shift
~	one's complement (unary)

A full discussion of bitwise arithmetic would be beyond the scope of this language reference. For those who are interested, any introductory book on the C programming language would be a suitable reference. The information below will be sufficient to remind those with prior experience of the function of each operator:

& Sets a bit to 1 in the result if and only if both of the corresponding bits in its operands are 1, and to 0 if the bits differ or both are 0. Example: 9 & 1 yields 1.

| Sets a bit to 1 in the result if one or both of the corresponding bits in its operands are 1, and to 0 if both of the corresponding bits are 0. Example: 9 | 2 yields 11.

^ Sets a bit in the result to 1 when the corresponding bits in its operands are different, and to 0 when they are the same. Example: 7 ^ 4 yields 3;

<< Shifts the first operand the number of bits to the left specified in the second operand, filling with zeros from the right. Example: 2 << 3 yields 16.

>> Shifts the first operand the number of bits to the right specified in the second operand, discarding the bits that 'fall off' at the right. Example: 34 >> 2 yields 8.

~ Inverts each bit in the operand, changing all ones to zeros and all zeros to ones. Example: `~0xFFFF0000` yields `0x0000FFFF`.


The large majority of users will never find a need for bitwise arithmetic, but it has been included in the interest of completeness.

Operator Precedence

The following table summarizes operator precedence and order of evaluation for the various operators supported by Boxer's Macro Language. Operators with the strongest/highest precedence are listed first:

Operator	Evaluates
( ) [ ]	left to right
! ~ ++ -- -	right to left
* / %	left to right
+ -	left to right
<< >>	left to right
< <= > >=	left to right
== != ~=	left to right
|	left to right
&	left to right
^	left to right
&&	left to right
||	left to right
? :	right to left
= += -= etc.	right to left
,	left to right

Parentheses can be used when required to ensure that the order of evaluation occurs as desired. For example:

```
n1 = 3 * 5 + 4;
```

assigns `19` to `n1`, while:

```
n1 = 3 * (5 + 4);
```

assigns `27` to `n1`.


Because the assignment operator (=) is evaluated from right to left, a construction such as the following is possible:

```
int i, j, k;
i = j = k = 0;
```

k is assigned the value 0, j is assigned the value of k, and i is assigned the value of j.

Character Constants

Boxer's Macro Language recognizes the standard character constants which have been popularized by the C programming language:

Sequence	Meaning	Decimal Value
'\b'	Backspace	8
'\f'	Formfeed	12
'\n'	Newline	10
'\r'	Carriage Return	13
'\t'	Tab	9
'\\'	Backslash	92
'\''	Single Quote	39
'\"'	Double Quote	34
'\0'	Null	0

In addition, Boxer will recognize a backslash (\) followed by three *octal* digits as the character whose ASCII value is given by the digits used. For example, `'\101'` could be used to represent a capital `A`, since its ASCII value, in octal, is 101.

Character constants can be used in any place that a `char` data type is expected, or within a double-quoted string: `"this is a string with a newline at the end.\n"`

Numeric Constants

Numeric `int` constants can be specified in either *decimal* or *hexadecimal* format:

```
int n1 = 32;
int n2 = 0x20;
```

Each of these assignments supplies the value `32` to `n1` or `n2`.

Numeric `float` constants can be specified in any of the following forms:

```
float x1 = 500;
float x2 = 500.0;
float x3 = 5e2;
```

```
float x4 = 5e02;
float x5 = 5.0e2;
float x6 = 5.0e02;
float x7 = 5.0e+2;
float x7 = 5.0e+02;
```

Each of these assignments results in the value 500 being assigned to the variable being declared.

For floating point values less than 1, the minus sign can be used to designate exponentiation. All of the following examples represent the number .05:

```
.05
0.05
5e-2
5e-02
5.0e-2
5.0e-02
```

Symbolic Constants

The following symbolic constants are recognized:

Name	Value
TRUE	1
FALSE	0
YES	1
NO	0
ON	1
OFF	0

These constants can be used in place of the values 0 and 1 to make a macro more readable. For example, you can write:

```
ViewBookmarks(ON);
```

instead of:

```
ViewBookmarks(1);
```

Declaring Variables

Variable names can be up to 32 characters in length and must not conflict with the names of any keywords or internal functions. Variable names can use alphanumeric characters and the underscore (_), but they must not start with a digit. All variables must be declared before use. Initialization of variables can be done at declaration-time, but this is not required. Uninitialized variables will be zero-filled automatically.

Boxer's Macro Language supports a flexible syntax for declaring variables. All of the following examples are legal declarations when they appear at the top of a macro, before other executable statements:

```
string s1;
string s2 = "Boxer";
string s3, s4, s5;
string s6 = "abc", s7, s8 = "def";

char c1;
char c2 = 'A';
char c3, c4, c5;
char c6, c7 = 'x', c8;

int n1;
int n2 = 10;
int n3, n4, n5;
int n6, n7 = -4, n8;

float x1;
float x2 = 1.05;
float x3 = 1.2e04;
float x4, x5, x6;
float x7, x8 = 7.75, x9;
```

In the spirit of the C programming language, Boxer's macro language also allows a `string` variable to be declared as an array of characters. The declaration:

```
char str[100];
```

is (for most purposes) functionally equivalent to the declaration:

```
string str;
```

for declaring a variable which can hold a short string of characters. See the *String Subscripting* section below for details on when the former style might be required.

Conditional Statements

Boxer's Macro Language supports three different conditional statements: `if`, `if-else` and the ternary statement. An `if` statement will be executed if the expression in parentheses evaluates to a non-zero result. Below are examples of the three conditional statements:

```
if (LineCount() > 10000)
   {
   longfile = true;
   }


if (LineCount() > 10000)
   {
```

```
   longfile = true;
   }
else
   {
   longfile = false;
   }


longfile = (LineCount() > 10000) ? true : false;
```

In the first example, the variable `longfile` is set `TRUE` if the return from the function `LineCount()` is greater than `10000`. In the second example, an `if-else` statement is used to additionally set `longfile` to `FALSE` if the condition is *not* met.

The final example illustrates the ternary statement, and its effect is identical to the `if-else` example immediately above it. If the condition within parentheses evaluates to `TRUE`, the expression immediately following the `?` is evaluated. If not, the expression after the `:` is evaluated. A ternary statement is effectively a compact `if-else` statement.


The ternary statement in Boxer's Macro Language is modeled after that of the C programming language, with one exception. In Boxer macros, the parentheses around the conditional expression are *required*, in C these parentheses are optional.


When a single statement is conditional upon an `if` or `if-else` statement, as is shown in the examples above, the use of curly braces `{ }` is not required. Curly braces *are* required when two or more statements are to be conditionally executed, or when those statements are the subject of a looping statement.

Looping Statements

Boxer's Macro Language supports three different looping statements: `for`, `while` and `do-while`. A loop statement will continue looping so long as the 'test' expression in parentheses evaluates to a non-zero result. Below are examples of each of these statements:

```
// find the longest line in the file
for (line = 1, longest = 0; line <= LineCount(); line++)
   if ((n = LineLength(line)) > longest)
         longest = n;


// find the longest line in the file
line = 1;
longest = 0;
while (line <= LineCount())
   {
   if ((n = LineLength(line)) > longest)
         longest = n;
```

```
   line++;
   }


// find the longest line in the file
line = 1;
longest = 0;
do
   {
   if ((n = LineLength(line)) > longest)
        longest = n;

   line++;
   }
while (line < LineCount());
```

The three loops above are functionally equivalent to one another, with one exception that will be discussed below.

The `for` loop is the most compact, since it permits the three elements of a loop's control to be specified on a single line: the initialization, the test, and the increment. These are found within the parentheses of the `for` loop and are separated by semi-colons. When a `for` loop is first executed, the initialization section is performed, and the test section is evaluated. If the test evaluates to a non-zero result, the statement(s) in the body of the loop are processed. At the end of the loop, the increment section is processed. Control then passes again to to the test section, to the body, and so on.


Boxer's Macro Language supports a very flexible `for` loop structure. The initialization, test and increment sections are each optional. Moreover, multiple initializations can be performed by separating the statements with the comma operator.

The `while` loop is a simpler loop, in that the only required control element that must be supplied is the test. For illustration purposes, the `while` loop above was written to be identical in function to the `for` loop above it. In fact, every `for` loop can be written as a `while` loop, and every `while` loop can be written as a `for` loop. A `for` loop is typically used when one needs to initialize and increment a loop index. A `while` loop is typically used when a single condition is sufficient to control the flow of the loop.

A `do-while` loop is essentially an upside-down `while` loop. A `do-while` loop tests at the bottom, whereas a `while` loop tests at the top. A `do-while` loop should be used in those cases where the loop is always to be be executed at least once. That leads us to why the `do-while` example above is not exactly equivalent to the `for` and `while` loops above it. If the current file is empty, the `for` and `while` loops above will not be executed. The `LineCount()` function will return `0` and the initial test will fail. In the `do-while` loop, the `LineCount()` call isn't made until the bottom of the loop. In the case of an empty file, the body of the loop would be processed and the `LineLength()` call would fail because the `line` parameter would

be out of range.

Sometimes the need arises to construct a 'forever' loop; one which will run until some condition within the body of the loop is satisfied and a `break` statement is executed. Both the `for` and `while` loops can be used for this purpose. Here are two examples:

```
// loop until the user enters the right answer
for (;;)
   {
   GetString("What's the capital of Arizona?", answer);

   if (answer ~= "Phoenix")
        break;
   }

// loop until the user enters the right answer
while (TRUE)
   {
   GetString("What's the capital of New Hampshire?", answer);

   if (answer ~= "Concord")
        break;
   }
```

Notice that these examples used the `~=` operator to ensure that the user's response was not rejected due to improper case.

Alert readers might notice that the above examples could be more neatly implemented using a `do-while` loop, since this is a case where the loop always wants to be run once, and the test can be more logically placed at the bottom of the loop:

```
do
   GetString("What's the capital of California?", answer);
while (strcmpi(answer, "Sacramento") != 0);
```

This example uses the `strcmpi()` function to perform a case insensitive string comparison, because the `~=` operator does not have a companion *string-does-not-match* operator.

The break Statement

The `break` statement can be used to exit from a loop prematurely. Control passes to the next statement following the loop which has been exited. For example:

```
// loop on all lines in the file
for (i = 1; i <= LineCount(); i++)
   {
   // exit the loop if a line is longer than 1000 characters
   if (LineLength(i) > 1000)
        break;
```

```
   }

// control passes to here after break
New;
```

The continue Statement

The `continue` statement can be used to jump to the bottom of a loop prematurely. Control passes to an imaginary label at the end of the loop. For example:

```
// loop on all lines in the file
for (i = 1; i <= LineCount(); i++)
   {
   // exit the loop if a line is longer than 1000 characters
   if (LineLength(i) > 1000)
        continue;

   // ... other processing ...

   // continue jumps to here
   }
```

The goto Statement

The `goto` statement can be used to jump unconditionally to a label. Control passes to the next statement after the label. For example:

```
// loop on all lines in the file
for (i = 1; i <= LineCount(); i++)
   {
   // exit the loop if a line is longer than 1000 characters
   if (LineLength(i) > 1000)
        goto toolong;

   // ... other processing ...

   toolong:
   // goto jumps to here

   // ... other processing ...
   }
```

The return Statement

The `return` statement can be used to end a macro prematurely. If a return statement is not encountered, a macro will run until the closing curly brace in the body of the macro is encountered.

Function Calls

Boxer's Macro Language includes a wide variety of functions that provide access to the editor's commands, configuration settings, and to string and math libraries. The

function set is documented in the Macro Function Reference, as well as in the Macro Dialog itself.

When making a function call, care should be taken to ensure that the parameters supplied to the function match the declared type(s) that the function expects to receive. Boxer is able to trap missing and/or mismatched parameters in most cases, but unexpected results can occur when invalid parameters are supplied.

Function names are not case sensitive; Boxer will accept function names that do not match the function name with regard to character case.

If a function does not require parameters, it is not necessary to supply parentheses at the end of the function name. For example:

```
LineCount();
```

and

```
LineCount;
```

are functionally equivalent, because the `LineCount` function does not require any parameters. That said, the practice of using `()` on all function calls can help to distinguish function names from variable names.

Simple expressions can be supplied to in a function call without difficulty, and they will be evaluated as expected before being sent to the function for processing. For example:

```
max(3 * 45, 4 * 90);
```

is a legitimate construction that might be used in calling the `max()` function. If you find that you are getting unexpected results in a case like this, introduce a temporary variable to hold the value of the expression, and then supply the variable to the function in place of the expression.

String Subscripting

Arrays are not supported in the classical sense; it's not possible to declare an array of `int` or `float` variables, for example. But Boxer's Macro Language does recognize a `string` variable to be an array of elements of type `char`, and allows those elements to be accessed individually through the use of subscripts. The first character within a string is located at index `0`, the second character is at index `1`, etc. In the following example:

```
string str = "BOXER";
char c1;
c1 = str[2];
```

the character variable `c1` would be assigned the value `'X'`.

Likewise, a `string` variable can be modified by assigning individual elements within

the string using subscripting:

```
string s1 = "water";
s1[0] = 'w';
s1[1] = 'i';
s1[2] = 'n';
s1[3] = 'e';
s1[4] = '\0';
```

This code fragment has the effect of changing the content of string variable `s1` from `"water"` to `"wine"`. Notice that the null character (`'\0'`) was used to shorten the string from five characters to four.

String subscripting makes it possible to use a string variable in the way that an array might be used. Here's an example that totals the number of occurrences of each letter within an input string:

```
macro array_example()
{
int i;
string input = "now is the time for all good men to come to the
aid of their country.";
char tally[256];        // note that all elements are initially
set to zero

// loop to process all characters in the input string
for (i = 0; input[i] != '\0'; i++)
   tally[input[i]]++;

// open a new, untitled file
New;

// report the results for lowercase letters
for (i = 'a'; i <= 'z'; i++)
   printf("letter %c occurred %d time(s)\n", i, tally[i]);
}
```

Had the `tally` array been declared as a `string` type, Boxer's built-in range checking would have prevented the string from being used in the way that was shown above. By declaring the string as a character array of sufficient size, the macro processor is forewarned that the code may later index into the string beyond the terminating null character.

Due to the capacity of the `char` data type (0-255), the utility of the above technique is limited to applications in which the maximum number of occurrences would be less than 256.

Type Conversions

Boxer's Macro Language will automatically convert between data types whenever possible in order to resolve an expression that involves mismatched data types. Here are some examples:

```
   string s1 = 'A';          // result: s1 gets "A" (char to string)
   string s2 = 65;           // result: s2 gets "A" (int to string)
   string s3 = 65.0;         // result: s3 gets "A" (float to string)

   char c1 = 65;             // result: c1 gets 'A' (int to char)
   char c2 = "A";            // result: c2 gets 'A' (string to char)
   char c3 = 65.0;           // result: c3 gets 'A' (float to char)

   int n1 = 'A';             // result: n1 gets 65; (char to int)
   int n2 = "123";           // result: n2 gets 123 (string to int)
   int n3 = 123.45;          // result: n3 gets 123 (float to int)

   float x1 = 'A'            // result: x1 gets 65.0 (char to float)
   float x2 = 65;            // result: x2 gets 65.0 (int to float)
   float x3 = "123.45";      // result: x3 gets 123.45 (string to
float)
```

Comments

Comments can be placed throughout a macro to help document the code. Two types of comments are supported, block comments and end-of-line comments:

```
/* this is a multi-line
   block comment */

int n1 = 7;          // this is an end-of-line comment
```

6.18 Main Menu

With the exception of the Cursor Movement commands, Boxer's main menu provides access to all of the editor's features and commands. Commands are grouped by function into ten top-level menus: File, Edit, Block, Search, Paragraph, Tools, Configure, View, Window and Help.

The main menu can be accessed with the mouse or by depressing *Alt* along with the underlined hot letter for the desired menu. Once a menu has been dropped, pressing a command's hot letter (with or without *Alt*) will execute the command.

Many commands also have *shortcut keys*, which provide a means to issue a command without entering the main menu structure. A command's active shortcut key--when available--is displayed to the right of that command's menu entry. The Configure Keyboard command can be used to change shortcut keys or to add secondary key assignments.

While navigating the main menu, issuing the Help command (which is assigned to *F1* by default) will display the help topic for the highlighted menu entry. This permits the main menu to be used as an index into the help system.

By default, icons are displayed next to many main menu entries. Studies have found that icons help many users to recognize and locate menu items and buttons more

quickly. The display of main menu icons can be controlled on the Configure | Preferences | Display options page. The option is titled *Display icons in menus*.

Several commands which are most likely to be used during editing are available on the Context Menu. The context menu is activated by clicking the right mouse button within an editing window.

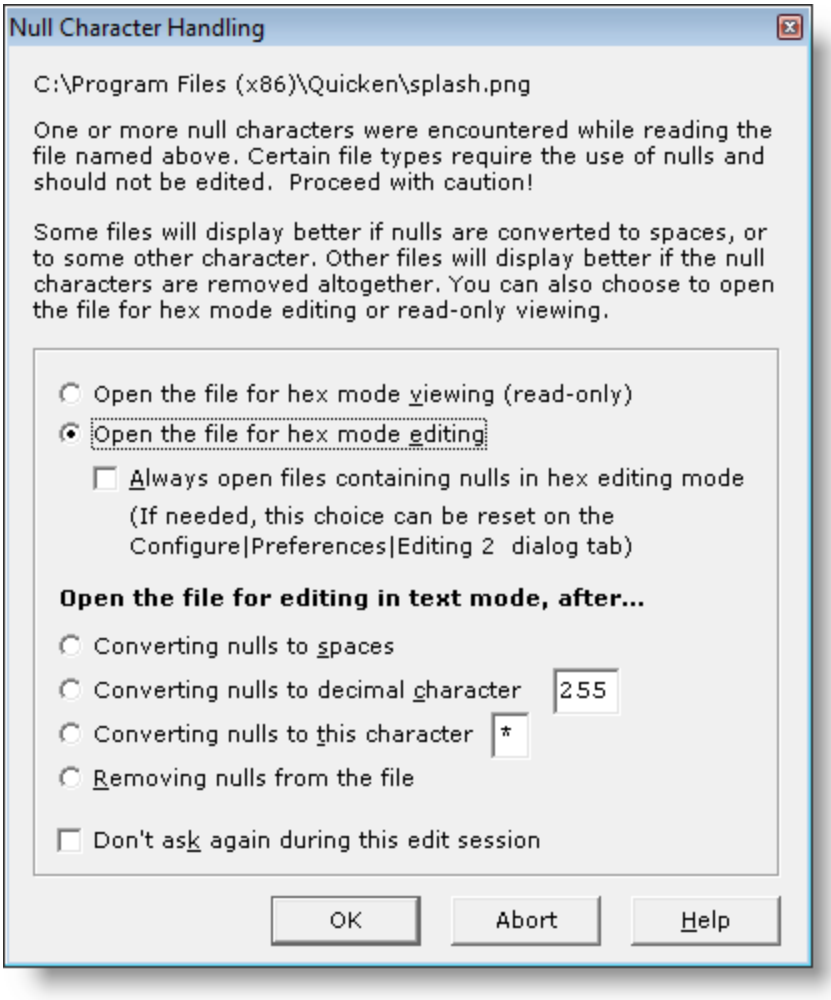
6.19 Null Characters

The term *text file* is generally considered to refer to those files that do not contain null characters (ie, the character whose ASCII value is zero), and which have periodic line enders to delimit the ends of lines. Binary or Hex files are those files which do not conform to these requirements.

When Boxer opens a text file, it does so using a conventional display format where each character occupies one position on screen, and newline characters are interpreted to cause a new line to be created. When Boxer opens a hex file, it uses a special display format where each character is represented by a two-digit *hexadecimal* code.

There may be times when you would like to edit a hex file using a conventional text file display. This situation may arise when a text file has become corrupt and now contains unwanted null characters, or when you're trying to salvage text from a file whose data format includes null characters, and you don't want to (or can't) open the file with its native application.

For these situations, Boxer has a Null Character Handling dialog with options for how null characters should be converted:



☞ An option appears on the Configure | Preferences | Editing 2 dialog page that allows you to indicate whether files containing nulls should always be opened in hex editing mode, or whether the Null Character Handling dialog should be displayed instead.

With this dialog you can control how null characters will be handled in the file you are loading. How you decide to proceed will depend on the type of file you're editing. It might be wise to first create a backup copy when editing files of this sort.

6.20 Portable Editing

Boxer can be run as a portable application: it runs seamlessly from removable media--such as a USB stick or Secure Digital card--and no 'footprints' are left on the host machine on which it is run. Boxer's installer provides an option to install to removable media, without creating a program group, icons, file associations, etc. on the host machine. By default, instead of writing its settings to the Windows registry, Boxer saves them to a disk-based file (named `BOXER.INI`) which resides in its data directory.

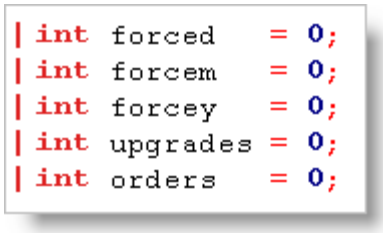
By using the -I command line option flag, an alternate INI file can be loaded. This makes it possible to maintain a virtually unlimited number of distinctly configured Boxer profiles.

An option to write to the Windows registry is available on the Configure | Preferences | Other dialog page.

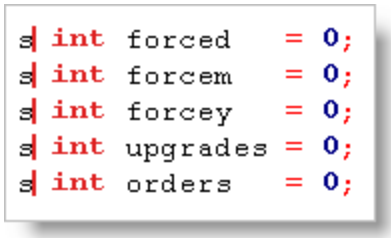
6.21 Power Columns

Boxer's Power Columns feature can be a big time saver when editing text that requires identical changes to be made on each line. In Power Columns mode, the text you type is applied to every line within the range of selected lines. When you cursor left or right, the insertion point moves in all selected lines. If you press *Delete*, or *Backspace*, a character is deleted in each line. You can even Paste a short text string into each line with just a single Paste command. Power Columns can work on two lines, two thousand lines... or even more.

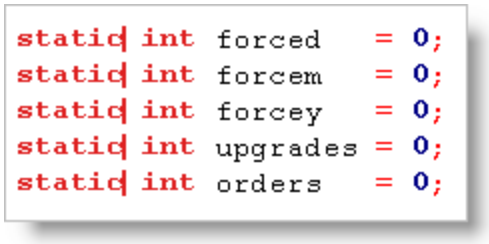
Let's consider an example... Suppose you've got five variable declarations that need to have the word 'static' applied to each of them. This is an editing task that arises in programming, but you'll probably be able to imagine other uses as well. To enter Power Columns mode, you simply need to create a Columnar Selection of zero width. Make sure the selection mode is set to Columnar on the Block menu, and then press *Shift+Down* four times. This will be the result:



The special red cursor bars indicate that Power Columns mode is active. When you press the letter 's', the character is inserted on each line:



When you type 'tatic' the rest of the text is entered, and the job is done:



(The word 'static' changed to red because 'static' is a reserved word.)

If you make a mistake while typing, the *Delete* and *Backspace* keys operate predictably to correct your error. If additional changes are needed in another portion of the line, the *Left* and *Right* arrow keys can be used to move the insertion point while still remaining in Power Columns mode. To exit Power Columns mode, press *Escape*, or use the *Up* or *Down* arrow.

6.22 printf and sprintf Formatting

Boxer's Macro Language has two functions that support formatted printing. `printf` and `sprintf` are functions that allow output to be formatted before being sent to a destination. For `sprintf`, the destination is a supplied `string` variable. For `printf`, the destination is the currently edited text file.

☞ The implementation of these functions is very similar to their implementation in the C programming language. Just a few of the most esoteric formatting options have been left unimplemented in Boxer's Macro Language. If you are already familiar with the formatting offered by `printf` and `sprintf`, you will probably not need to consult this reference section.

The formatting of the output is controlled by a *format string*. The format string is a character string containing two types of objects: ordinary characters which are copied directly to the destination, and *conversion specifications*, each of which is introduced with the `%` symbol.

A conversion specification has the following form:

```
%[flags][width][.precision][type]
```

The fields of the conversion specification have the following meanings:

flags (optional)

`-`
Left-justifies the result, pads on the right with blanks. If not supplied, it right-justifies the result, padding on the left with zeros or blanks.

`+`
Signed conversion results will always begin with a plus (+) or minus (-) sign.

space
If the value is non-negative, the output begins with a space instead of a plus; negative values begin with a minus.

width (optional)

n
At least *n* characters are printed. If the output value has fewer than *n* characters, the output is padded with spaces.

0*n*
At least *n* characters are printed. If the output value has fewer than *n* characters, it is filled on the left with zeros.

precision (optional)

```
(none)        Precision set to default:
                1 for d, i, o, u, x, X types
                6 for e, E, f types
                all significant digits for g, G types
                print to first null character for s types
                No effect on c types

.0            For d, i, o, u, x types, precision is set to default
              For e, E, f types, no decimal point is printed.

.n            n characters or n decimal places are printed.
               If the output value has more than n characters, the
output might be truncated
               or rounded. (Whether or not this happens depends on
the type character.)
```

No numeric characters will be output for a field (i.e., the field will be blank) if the following conditions are all met:

- you specify an explicit precision of 0
- the format specifier for the field is one of the integer formats (**d**, **i**, **o**, **u**, or **x**)
- the value to be printed is 0

How *precision* affects the conversion performed for each type:

```
Type          Effect of precision (.n) on conversion
-------------------------------------------------------------------
-----------------------
 d            Indicates that at least n digits are printed. If input
argument has fewer than
 i           n digits, output value is left-padded x with zeros. If
input argument has more
 o           than n digits, the output value is not truncated.
 u
```

x
X

e Specifies that *n* characters are printed after the decimal point, and the last
E digit printed is rounded.
f

g Specifies that at most *n* significant digits are printed.
G

c Has no effect on the output.

s Specifies that no more than *n* characters are printed.

type (required)
The *type* parameter specifies what kind of conversion `printf` or `sprintf` performs. The following conversion characters are supported:

%
Prints the percent character (`%`).

c
Prints a single character.

s
Prints a string.

d
Prints a signed decimal integer.

i
Prints a signed decimal integer (same as **d**)

o
Prints a signed octal integer.

u
Prints an unsigned decimal integer.

x
Prints an unsigned hexadecimal integer using `a-f` as may be required.

X
Prints an unsigned hexadecimal integer using `A-F` as may be required.

f
Prints a floating point value of the form `[-]9999.9999`.

e
Prints a floating point value of the form `[-]9.9999e[+|-]999`.

E
Prints a floating point value of the form `[-]9.9999E[+|-]999`.

g
Prints a signed value in either **f** or **e** form, based on the value and precision. Trailing zeros and the decimal point are printed only if necessary.

G
Prints the same as **g**, but uses 'E' for the exponent if an exponent is needed.

Example Format Strings

```
format string          output
-------------------------------------------------------------------------
------------------------
"save 25%%"            save 25%
"%-25s"                outputs a string left justified in a
25-character wide field
"%8d"                  output a decimal value right justified in an
8-character wide field
"%10.2f"               outputs a floating point value with 2
decimal places in a 10-character
                       wide field
"%08X"                 output a hexadecimal value with leading
zeros in an 8-character wide
                       field
"\"yes\""              "yes"
```

Example Macro

This macro illustrates the use of `printf` with a formatting string that includes the `%c`, `%d`, `%x` and `%o` format specifiers. The decimal value (`%d`) will be right justified in a field of three characters. The hexadecimal value (`%x`) will be printed in a field of three characters with a leading zero. The octal value (%o) will be printed in a field of three characters with leading zeros. Notice that the variable `i` is placed on the argument list once for each format specifier that appears in the format string.

```
macro chart()
{
// open a temporary new file
New;

// loop from the space character to value 255
for (int i = ' '; i <= 255; i++)
   printf("char: '%c'   dec: %3d  hex: %02X  oct: %03o\n", i, i, i,
i);
}
```

6.23 Regular Expressions

When searching for a text string using the Find, Replace, Replace Line Enders or Find Text in Disk Files commands, Boxer supports the use of Regular Expressions, a pattern matching grammar first popularized on the Unix operating system. Regular Expressions make it possible to specify a search string which can match many different target strings, or to restrict the ways in which a search string can be matched.

Boxer uses Perl-Compatible Regular Expressions as implemented by the increasingly popular PCRE 5.0 library. See the end of this topic for further information and acknowledgements.

A complete treatment of the topic of regular expressions could--and does--fill an entire book. *Mastering Regular Expressions*, by Jeffrey Friedl is one such book, and a good one at that. This help topic was written to acquaint the typical user with the most common regular expression features, without getting too bogged down in fine details. The advanced reader is encouraged to seek out additional information on the web, or within the PCRE documentation itself. We have posted one such reference document on our site for your convenience.

Regular Expressions are *very* powerful, and can be more easily understood by studying several examples.

Matching a Single Character
The dot (.) will match any single character, except the newline character. Example: `p.t` will match `pat`, `pet`, `pit`, `pot`, and `put`, and in fact any 3-character sequence with `p` and `t` at its ends and a single character in the middle.

Matching with an Asterisk
The asterisk (`*`) will match zero or more occurrences of the preceding character. Example: `zo*m` will match `zm`, `zom`, `zoom` and `zooooooooom`, among others. Note that the character preceding the asterisk can be the dot, so zero or more occurrences of *any* character will be matched when the construction `.*` is used. Example: `Bo.*r` will match `Boxer`, `Bowler`, `Bookmaker`, `Bookkeeper` and `Building Manager`.

Matching with a Plus Sign
The plus sign (`+`) will match one or more occurrences of the preceding character. Example: `ho+p` will match `hop`, `hoop` and `hooooooop`, among others. Note that the character preceding the plus sign can be the dot, so that one or more occurrences of *any* character will be matched when the construction `.+` is used.


Patterns that use either `*` or `+` can often result in more than one possible matching string. This concept is known as minimal or maximal matching. You can control whether Boxer will return the shortest or longest matching string using the *Maximal matching* checkbox on any dialog where regular expressions are permitted.

Matching at Start of Line
The caret (`^`) can be used to force a match to occur at the start of a line. Example: `^The` will match any line beginning with `The`.

 You can also force a start-of-line match using the checkbox provided on the dialog.

Matching at End of Line
The dollar sign (`$`) can be used to force a match to occur at the end of a line. Example: `result$` will match any line ending with the word `result`.

 You can also force an end-of-line match using the checkbox provided on the dialog.

Character Classes or Range Expressions
One or more characters can be placed within square brackets to designate the characters which can match in that position. Example: `p[aeiou]t` will match `pat`, `pet`, `pit`, `pot` and `put`. Note that digits are also characters, so an expression such as `201[1234]` will match any of `2011`, `2012`, `2013` or `2014`.

Characters can also be placed within square brackets with a dash between them to designate a range of characters. Example: `[b-d]ent` will match `bent`, `cent` and `dent` because the expression `[b-d]` is shorthand for all characters in that range. The character range can be entered in ascending or descending order; both `[A-Z]` and `[Z-A]` are allowed and are functionally equivalent.

The character set appearing within square brackets can be negated by using the caret (`^`) as the first character within the opening square bracket. Example: `[^cb]ent` will match `tent`, `rent`, `sent`, `dent` and others, but *not* `cent` or `bent`. The caret can also be applied to negate a character range within square brackets: `[^a-e]` will match all characters *except* `a`, `b`, `c`, `d` and `e`. If the caret appears anywhere else within the range expression, its meaning reverts to that of matching the caret itself.

Matching Multiple Strings
The vertical rule (`|`) can be used to separate two or more regular expressions so that any of the patterns will match. Example: `red|green|blue|yellow` will match any of the color names that are separated by the vertical rules.

Subpatterns
Left and right parentheses can be used to start and end a *subpattern*. Example: `c(ar|en|oun)t` will match `cart`, `cent` and `count`. In absence of the parentheses, `car|en|ount` would match `car`, `en` or `ount`... a very different result.

Escape Character
The backslash can be used to remove significance from a pattern matching character. Example: if you need to search for an asterisk, use `\*`. To search for a dot, use `\.`. To search for a plus sign, use `\+`. To search for the backslash itself, use `\\`.

 You can also remove significance from pattern matching characters by placing them inside a range expression. For example, `[*+]` could be used to match either an asterisk or a plus sign.

Matching Whole Words
To force a pattern to find only those occurrences of a search string which appear as

whole words, the pattern can be surrounded with a sequence that forces a match at a word boundary. Example: to find the word `sign`, but not words such as `assign`, `signature` or `assignment`, use `\bsign\b`.

 You can also force a whole word match using the checkbox provided on the dialog.

Matching Special Characters
Several characters that are not readily typed from the keyboard can be matched using special character sequences:

`\\`	match a backslash character
`\a`	match a bell (alarm) character (ASCII 7)
`\b`	match a backspace character (ASCII 8) (only if used in a character class)
`\cx`	match character Control-x (x = any character)
`\e`	match an escape character (ASCII 27)
`\f`	match a formfeed character (ASCII 12)
`\n`	match a newline character (ASCII 10)
`\r`	match a carriage return character (ASCII 13)
`\t`	match a tab character (ASCII 9)
`\ddd`	match *octal* character ddd (d = any digit 0-7)
`\xhh`	match *hexadecimal* character hh (h = any hex digit)

Generic Character Types
There are several convenient shorthand sequences for matching common character classes:

`\d`	match a decimal digit (0-9), equivalent to: `[0-9]`
`\D`	match any character except a decimal digit, equivalent to: `[^0-9]`
`\s`	match any whitespace character, equivalent to: `[\t\n\f\r ]`
`\S`	match any character except whitespace, equivalent to `[^\t\n\f\r ]`
`\w`	match any word character, equivalent to: `[_a-zA-Z]`
`\W`	match any character except a word character, equivalent to: `[^_a-zA-Z]`

 A word character is considered to be any letter, digit or underscore. No consideration is made for accented characters that reside above value 128 in the character set. If you require such characters in a pattern, you'll need to name these characters explicitly, perhaps in a range expression that also uses `\w`.

Assertions
The following sequences can be used to force a match to occur only at a required position:

\b	match at a word boundary
\B	match when not at a word boundary
\A	match at start of subject
\Z	match at end of subject or before newline
\z	match at end of subject
\G	match at first matching position in subject

Useful Constructions
The following examples illustrate some common constructions, and give examples of the utility--and complexity--of some advanced regular expressions:

.*	match zero or more occurrences of any character
.+	match one or more occurrences of any character
^$	match an empty line
^\s+$	match a line containing only *whitespace*
^\s+	match leading whitespace
\s+$	match trailing whitespace
[a-zA-Z]	match any alphabetic character
this|that	match 'this' or 'that'
\b(\w+)\s+\1\b	match repeated words (such as 'the the')
\b[A-Z0-9._%-]+@[A-Z0-9._%-]+\.[A-Z]{2,4}\b	match a valid email address

Min/Max Quantifiers
A min/max quantifier can be used to control how many instances of the preceding entity are to be allowed within a match. The syntax for min/max quantifiers is summarized in this table:

{	start a min/max quantifier
}	end a min/max quantifier

{3}	match exactly 3 of the previous item
{3,}	match at least 3 of the previous item
{3,5}	match at least 3, but no more than 5 of the previous item

Example: the pattern [abc]{4,8} would match a sequence of characters consisting of the letters a, b or c, so long as at least 4 characters are present, and no more than 8 appear. Potential matches: aaaa, accb, abcabc, bbbbcccc. Non matches: aaa, abcd, abcabcabc.

☞ Careful readers might observe that * is effectively shorthand for {0,} and + is shorthand for {1,}.

Back References and Named Subpatterns
One of the more powerful features of Perl regular expressions is the ability to make reference within a pattern to the string that matched a subpattern which occurred earlier in the pattern. Subpatterns are created when a portion of a pattern is enclosed in left and right parentheses. The first opening left parenthesis encountered starts a subpattern whose number is 1. The second left parenthesis creates subpattern 2, and so on. To make a back reference to a subpattern by number, this syntax is used:

```
\1          back reference to subpattern number 1
```

Referring to subpatterns by number can get confusing when a complex regular expression is being created. For this reason, *named subpatterns* are also permitted. To start a subpattern named 'foo', the following syntax would be used:

```
(?P<foo>start a subpattern named 'foo'
```

Later on in the pattern, the string that matched subpattern 'foo' could be referenced using this syntax:

```
(?P=foo)back reference to the subpattern named 'foo'
```

The example presented above that matches repeated words used a back reference:

```
\b(\w+)\s+\1\b
```

The subpattern (\w+) matches any string that contains one or more word characters. In order for the entire pattern to match, that same string must appear again (due to the \1 reference) with one or more spaces (\s+) in between. Finally, the \b sequences at each end ensure that the pattern matches only at a word boundary.

☞ Named subpattern references can also be used in the replace string of the Replace and Replace Line Enders commands, and with the ChangeStringRE() macro function.

Closing Example
Finally, it's worth mentioning that any or all of the expressions presented above can be used within the same regular expression. This artificially complex example:

```
^The\sq[^a]ic{1}k.*f[aeiou]x.*ov[a-e]r.*lazy\040dog\.$
```

would match the sentence:

```
The quick brown fox jumped over the lazy dog.
```

so long as it appeared on a single line.

PCRE 5.0 License

The Perl-Compatible Regular Expression (PCRE) package used by Boxer was written by Philip Hazel, and is used in accordance with the PCRE license:

Copyright (c) 1997-2004 University of Cambridge
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of the University of Cambridge nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

6.24 Restoring an Edit Session

When Boxer is closed it records a wide variety of information about the current edit session. This information includes:

- Names of open files
- Cursor position in each file

- Bookmark locations
- Window positions, sizes and splits
- Text selection mode
- Visual Wrap mode
- Typing Wrap mode
- Read-only mode
- Hex mode
- Edit mode

If Boxer is later restarted without being given a filename to edit, it will automatically restore the last edit session. All of the files from the previous session will be restored, as well as the cursor position within each file.


Restoration of the previous session can be disabled with the *Always start with a new file (when no files are name)* option, found on the Configure | Preferences | Other dialog page.


If you would like to be able to open other file groups quickly, see the New Project topic for information about that feature.

6.25 Send-To Menu

During installation, Boxer's setup program provided the option to have Boxer installed on the *Send-To* menu. The Send-To menu appears as one of the entries in the *context menu* when a file's icon is clicked with the right mouse button. The Send-To menu provides a way to send a file to various programs (or devices) which are capable of receiving it. Depending on your work style, you may find it useful to be able to launch Boxer in this way. Note that if a copy of Boxer is already running, the new file(s) will be added to the existing Boxer session.

6.26 Sizes and Limits

Below is a table of sizes and limits which apply to various features and operations:

maximum number of open files/windows	no limit ①
maximum number of lines in a file	2,147,483,648 (2 GB)
maximum number of characters per line	32,767 (32 K)
maximum number of characters in a file	2,147,483,648 (2 GB)
maximum number of characters in a hex mode file	512,831,915
maximum length of a filepath, with its D:\ designator prefix	259 characters ②
maximum length of a filename, or filepath, less its D:\ designator	256 characters ②
minimum undo buffer size	2,048 (2 KB)

maximum undo buffer size	65,535 (64 KB)
minimum screen size	800 x 600
minimum recommended screen size	1024 x 768
minimum number of recent files on Recent Files submenu	0
maximum number of recent files on Recent Files submenu	24
maximum number of recent files on Recent Projects submenu	16
number of bookmarks	10
maximum length of a Fill String	512
number of internal clipboards	8
maximum size of an internal clipboard saved from session to session	2,048 (2 KB)
number of User Lists	8
maximum length of an item in a User List	256
maximum length of a User List title	40
number of User Tools	24
maximum length of a User Tool name	32
maximum length of a search string	512
maximum number of search strings saved in history list	20
maximum number of replace strings saved in history list	20
maximum number of Syntax Highlighting languages	100
maximum length of Syntax Highlighting language name	20
maximum line length within the Syntax.ini file	16,384
maximum number of reserved words in a language	3,000 ③
maximum number of template sets	100
maximum number of templates in a template set	500
maximum length of text in a template	2,048 (2 KB)
maximum length of a template name	100
maximum length of a template set name	32
maximum number of variables in a macro	200
maximum length of the 'string' data type	2,048 (2 KB)
maximum length of a macro variable name	32
maximum length of the FTP username field	40
maximum length of the FTP password field	40
maximum number of FTP accounts	100

① subject only to the virtual memory limits of the operating system

② these limits are imposed by the operating system

③ 3000 is approximate. The actual limit is governed by the maximum line length within the Syntax.ini file. The combined length of a comma-separated list of all reserved words within a given class (Reserved 1, Reserved 2, or Reserved 3) is limited to 16 KB.

6.27 Transferring Preferences

By default, Boxer records many of the editor's preferences and settings to a disk-based configuration file named `Boxer.ini`. This file contains all of the options within the multi-page Preferences dialog, as well as window sizes and positions, screen colors, fonts, private clipboard content, recent files, search strings, user tools and much, much more.

The rest of Boxer's settings are maintained in separate .INI and .TXT files in order to make them more readily editable and to facilitate exchange with other Boxer users. Here is a complete list of the on-disk configuration files and their content:

`Boxer.ini`	`General preferences and settings`
`Template.ini`	`Template information`
`Toolbar.ini`	`Toolbar information`
`Toolbar Icons\*.*`	`Custom toolbar icons`
`Syntax.ini`	`Color Syntax Highlighting information`
`BoxerCalc.ini`	`Calculator information`
`BoxerFilePicker.ini`	`File Picker information`
`Macros.ini`	`Macro variable storage`
`*.kbd`	`Keyboard layout files`
`*.bkr`	`Keystroke recording files`
`User List ?.txt`	`User List files`
`userdict.txt`	`User-defined dictionary entries`
`AC_words.txt`	`Auto-Complete dictionary`

If it becomes necessary to transfer Boxer from one PC to another, and you'd like to ensure that all of the old settings are maintained in the new installation, the above files should be copied to the new installation folder. Take care to maintain the directory structure; some of the files mentioned above reside in sub-folders within the data folder.

 The Explore Data Folder command can be used to quickly locate Boxer's data folder.

Windows Registry Option

Boxer also provides an option to store its settings to the *Windows Registry*. This option appears on the Configure | Preferences | Other dialog page. When this option is selected, the settings which would have been written to `Boxer.ini` are instead written to the Windows registry. Transferring preferences for this configuration then becomes a two-part process:

1. move the disk-based files named above.
2. move the Windows Registry key containing the Boxer settings.

To move information from the *Windows Registry*, use one the following procedures:

Windows XP, Vista and Windows 7:

- Run `REGEDIT.EXE`
- Locate and highlight the registry key: `HKEY_CURRENT_USER\Software\Boxer Software\Boxer Text Editor 14`
- From the *File* menu, select the Export command
- Save the registry key using a `.REG` file extension
- After copying the file to the destination PC use the *Import* command on the *File* menu to load the settings or double-click on the `.REG` file from within Explorer.

Windows NT/2000:

- Run `REGEDT32.EXE`
- Locate and highlight the registry key: `HKEY_CURRENT_USER\Software\Boxer Software\Boxer Text Editor 14`
- From the *Registry* menu, select the *Save Key* command
- Save the registry key using a `.REG` file extension
- After copying the file to the destination PC use the *Restore* command on the *Registry* menu to load the settings or double-click on the `.REG` file from within Explorer.

Windows 95/98/Me:

- Run `REGEDIT.EXE`
- Locate and highlight the registry key: `HKEY_CURRENT_USER\Software\Boxer Software\Boxer Text Editor 14`
- From the *Registry* menu, select the *Export* command
- Export the selected branch to a `.REG` file
- After copying the file to the destination PC use the *Import Registry File* command on the *Registry* menu to load the settings or double-click on the `.REG` file from within Explorer.

☞ In order to be sure the latest settings are stored in the Registry, make sure you exit Boxer before exporting the Registry information.

☞ On a networked PC, it may be necessary to login at a higher level to get access to RegEdit. It may then be necessary to login under the normal user name so that the

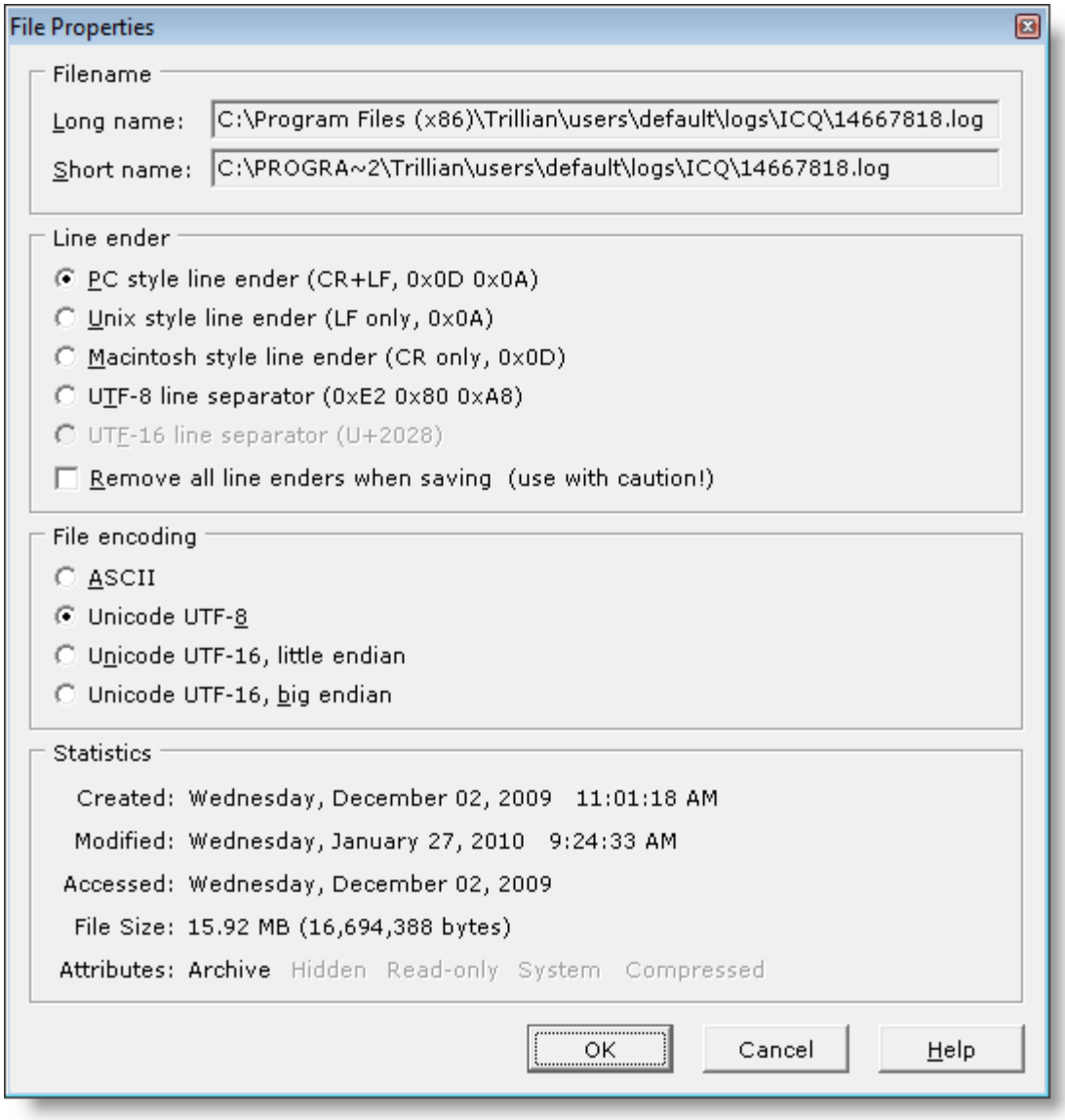
HKEY_CURRENT_USER entries will be visible.

☞ Some versions of RegEdit have a bug. If RegEdit reports trouble reading back a `.REG` file that it has created, one Boxer user has suggested saving the file as a `.HIVE` file instead.

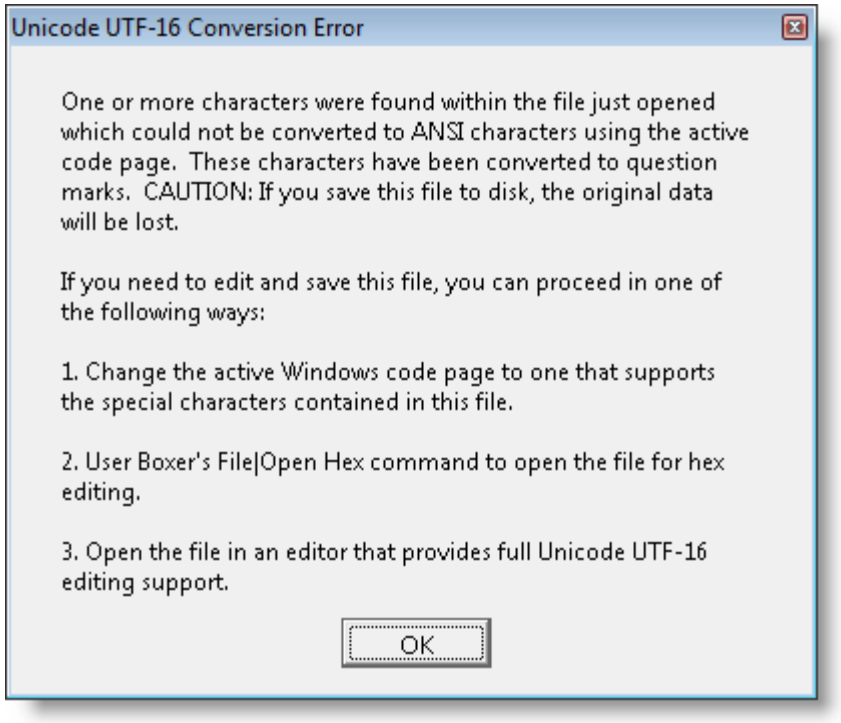
6.28 Unicode Files

Unicode is a standard for encoding text files so that all manner of international characters and symbols can be unambiguously represented in a text file, regardless of the file's source or destination computer system. There are several different Unicode file formats that are in popular use. Boxer can open and edit files which have been stored in the following formats: UTF-8, UTF-16 little endian, and UTF-16 big endian.

Once a Unicode file has been opened for editing, it will be treated like any other text file, and all editing operations are available for use on the file. If you would like to change the file encoding of a Unicode file -- or change an ASCII format file to Unicode format -- you can do so by visiting the File Properties dialog. It contains options that control what format will be used when the current file is saved to disk:



Due to its roots as an 8-bit ASCII text editor, Boxer's does not store Unicode files internally in 16-bit format. Rather, it converts Unicode files to ANSI format as they are opened by mapping characters onto the active *Code Page*. This enables Boxer to edit all Unicode files that contain text that is 7-bit in nature (such as .REG registry files, some XML files, Microsoft SQL files, etc.), as well as those UTF-8 and UTF-16 files with multi-byte content that resides within a single Windows code page. Boxer *cannot* be used to reliably edit Unicode files with content that is drawn from several code pages (for example, a file which contains text from many languages), or with content not present in the active Code Page. When a file cannot be converted to ANSI using the active Code Page, a dialog will appear to alert you to this, and will provide alternate options:



The active *Code Page* can be viewed using the System Info option on Boxer's About dialog.

When a Unicode file cannot be opened for editing in text mode, it can still be opened in Hex Mode.

6.29 Uninstalling Boxer

At the time of installation, Boxer's Setup program recorded information to enable the Uninstall utility to remove Boxer at a later time. There are two ways to run the Uninstall utility:

- From the *Start* menu select *All Programs* and the *Boxer Text Editor*. Within Boxer's submenu is an entry for the Uninstall utility.
- From the *Start* menu, select *Settings* and then *Control Panel*. Within the Control Panel group is an icon for *Add/Remove Programs*. Boxer will have an entry in the list of programs which can be removed.

7 Glossary

7.1 Glossary A-Z

A

S

short filename
shortcut key

T

task bar
thumb and scroll box
tool tip

U

URL

W

whitespace
WYSIWYG

7.2 acronym

An *acronym* is created by taking the first letter from each of a series of words and using these to create a new term. For example, *URL* is an acronym for *Universal Resource Locator*.

7.3 binary

A *binary* value is a number represented in base 2. The digits 0 and 1 are used when expressing binary values.

7.4 binary file

The term *binary file* refers to a file which contains Null characters. Binary files cannot be edited by Boxer.

7.5 client area

The *client area* is the area within a program's window which is used for the display of user documents.

7.6 code page

A code page is a character encoding table maintained by the operating system. A code page assigns numeric values ranging from 0 to 255 to a set of characters. In Windows, a code page typically contains common alphabetic characters and symbols in the lower half of the table (0-127), while the upper half of the table (128-255) contains a collection of accented or special characters used to represent non-English languages.

7.7 context menu

A *context menu* is a popup menu which is accessed by clicking on an object with the right mouse button. The context menu contains entries which are specific to the object

which was clicked upon.

7.8 data folder

Boxer's *data folder* is the folder/directory in which Boxer stores its configuration files. On Windows XP and previous versions, this is the same as the *program folder*. On Windows Vista and later, a special area is used for application data files.

7.9 decimal

A *decimal* value is a number represented in base 10.

7.10 file filter

A *file filter* is an expression which matches a class of files. For example, the file filter `*.txt` matches all files with a `.txt` extension. File filters can be used within dialog boxes to limit the file display to those which match a selected class of files.

7.11 fixed width

A *fixed width* font is one in which all characters in the font have the same width. For example, the letter 'i' will occupy just as much space as the letter 'w'.

7.12 focus

A control (or program) is said to have *focus* when it is the control which is receiving input from the keyboard.

7.13 footer

A *footer* is a line of text which appears at the bottom of a page, below the body text of the page. A footer might contain a page number, or the name of a chapter or section.

7.14 header

A *header* is a line of text which appears at the top of a page, above the body text of the page. A header might contain a chapter name, or the time and date that the document was printed.

7.15 header file

A *header file* is a file used by programmers to hold various declarations and definitions. Each source code file typically has an associated header file which contains the definitions relevant to that file. In the C++ programming language, the file `project.cpp` would use a file named `project.h` or `project.hpp` as its header file.

7.16 hexadecimal

A *hexadecimal* value is a number represented in base 16. The digits 0-9, and the letters A-F are used when expressing hexadecimal values.

7.17 hot letter

A *hot letter* is an underlined character which can be used to execute a menu item or control. Press *Alt* plus the hot letter to execute the item or control. If the hot letter appears in a menu which is already dropped, simply press the hot letter itself.

7.18 landscape

The term *landscape* refers to a paper orientation in which the long edge of the paper is placed horizontally.

7.19 long filename

A *long filename* is one which makes use of the more flexible file naming standards available under the 32-bit Windows operating systems. Filenames can be up to 256 characters in length, and may contain one or more embedded spaces.

7.20 maximal matching

The term *maximal matching* refers to a type of searching and reporting used by regular expression search routines. When maximal matching is performed, the longest matching string for a given regular expression will be reported as the matched string for a given line. In the lines below, the bold sections show the *minimal match* and the *maximal match* when the regular expression `o.*m` is applied to the sample text:

The quick br**own fox jum**ped over the lazy programmer.

The quick br**own fox jumped over the lazy programm**er.

7.21 modal and non-modal

A *modal* window or dialog box is one which retains *focus* until it is dismissed. A *non-modal* window or dialog box is one which can share focus with other open windows.

7.22 octal

An *octal* value is a number represented in base 8. The digits 0-7 are used when expressing octal values.

7.23 portrait

The term *portrait* refers to a paper orientation in which the long edge of the paper is placed vertically.

7.24 private clipboard format

A *private clipboard format* is a data format defined by an application program for storing information on the Windows clipboard. Using a private clipboard format permits an application to store complex data objects on the clipboard for later retrieval by that program. Because the format is private, other applications will not be able to access the clipboard data in the usual way. By contrast, when ordinary text is placed on the Windows clipboard, it is typically placed in a format which can be retrieved by all programs.

7.25 program folder

Boxer's *program folder* is the folder/directory in which Boxer is installed. By default, this is `c:\Program Files\Boxer Text Editor`, unless another directory was used at installation time.

7.26 proportionally spaced

A *proportionally spaced* font is one in which the display width varies according to the width of the character. For example, the letter 'i' will occupy less space than the letter 'w'.

7.27 short filename

A *short filename* is one which conforms to the naming conventions first popularized under MS-DOS. The filename portion can be up to 8 characters long, and the file extension can be up to 3 characters long. Spaces are not permitted anywhere within the filename or extension.

7.28 shortcut key

The term *shortcut key* refers to a key sequence which can be used to activate a menu command. For example, Ctrl+V is a shortcut key for the Paste command.

7.29 task bar

The term *task bar* refers to the area on the Windows desktop where active and inactive applications are displayed in icon form. Clicking on a button in the task bar will bring that application to the foreground.

7.30 thumb and scroll box

The terms *thumb* and *scroll box* refer to the draggable rectangle within a scroll bar which can be used to move through a document.

7.31 tool tip

The term *tool tip* refers to text which appears in a small popup window when the mouse is allowed to hover on top of a control.

7.32 URL

The term *URL* is an acronym which stands for Universal Resource Locator. An Internet address such as `http://www.boxersoftware.com` is a URL, as is `ftp://somedomain.com/somefile.zip`.

7.33 whitespace

Whitespace is a term used to refer collectively to the Space, Tab and Newline characters. While these characters do influence the look of a document, they are not normally visible on-screen.

7.34 Windows Registry

The *Registry* is a database maintained by Windows for its benefit and for the benefit of application programs. The Registry contains a wide variety of configuration information organized in a hierarchical structure of 'keys' and 'values'.

7.35 WYSIWYG

WYSIWYG is an acronym for What You See Is What You Get. It refers to a screen display which attempts to show a document as it will appear once printed.

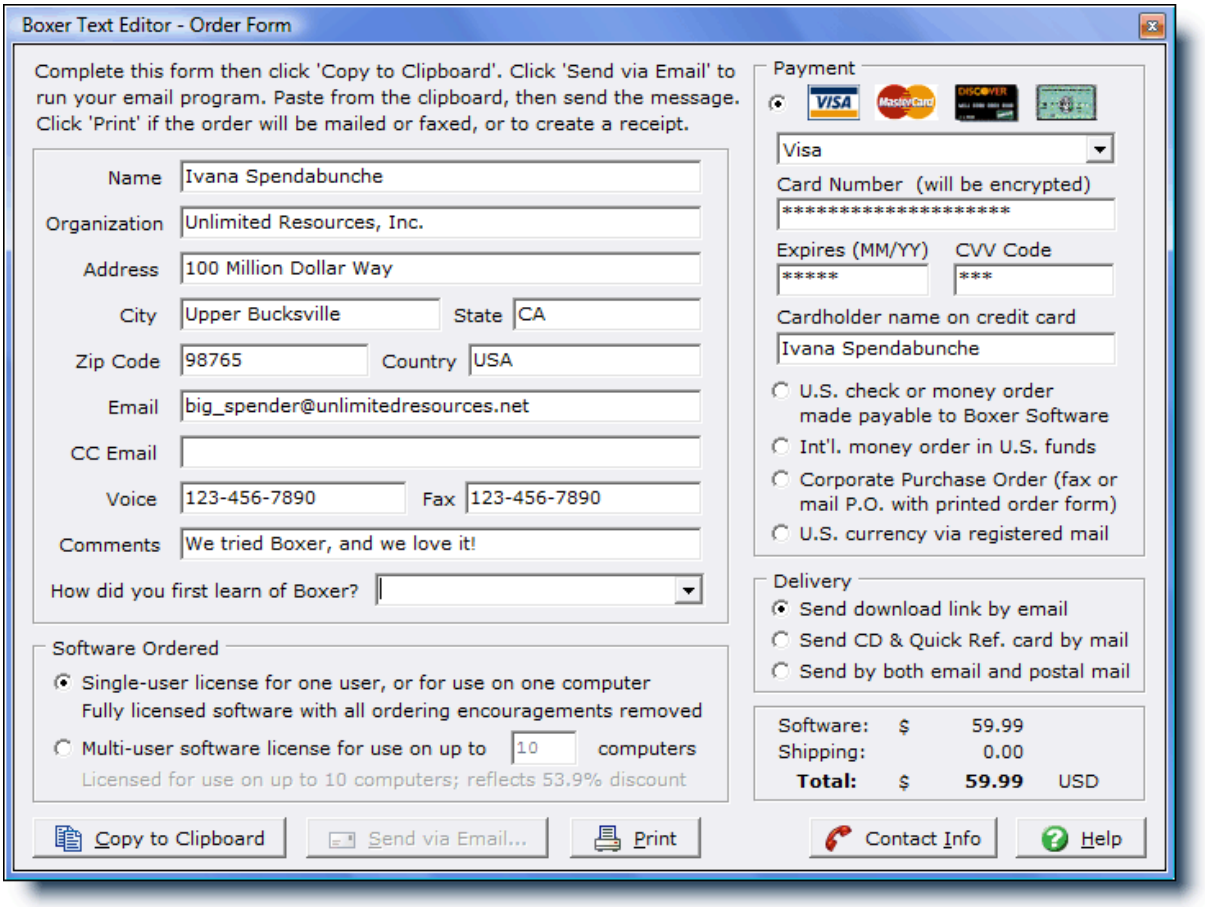
8 Ordering Boxer

8.1 Order Boxer

Menu: Help > Order Boxer

Default Shortcut Key: none

Boxer has an in-software order form to make ordering fast and easy. The order form is available by selecting the *Order Boxer* option from the Help menu, or by clicking the dollar bill icon on the toolbar of the evaluation version.



This order form can be used to submit your order by email, or to print an order form which can later be faxed or mailed along with payment. In all cases you can be assured that your order will receive prompt attention, and that we will safeguard your personal information. Boxer Software does not share its customers' mailing addresses, or email addresses, with any third parties.

If you prefer to print an order form from within this help file, and then mail or fax it to us, use this order form.

The Order Form will compute your total automatically as you complete your order. If a Multi-User License is being ordered, click the appropriate option and enter the quantity desired. The total will be updated automatically to reflect the quantity ordered. Likewise, shipping is computed according to the destination country, and depending on whether delivery will be made by email or postal mail (delivery by email is free). If you elect to have the software sent via email, an http link will be sent from which you can download the software; the software is not sent by email attachment.

Ordering by Email

Complete the form, and then click *Copy to Clipboard* to copy the information entered to the Windows clipboard. Click *Send via Email* to launch your email program. The *To*

field of your email program should auto-fill with sales@boxersoftware.com. Paste the order information from the clipboard into the message body and send the message in the usual way. Note that your credit card information will be encoded using a proprietary encoding algorithm for added security. We will decode the information after your order arrives.

☞ If your email program does not launch after clicking *Send via Email*, simply start it in the usual way and paste the content on the clipboard into the body of a new message. Send the message to sales@boxersoftware.com.

Ordering at our Website

Visit www.boxersoftware.com to order from our secure order page. Full ordering details are provided at the site.

Ordering by Phone

Call toll-free within the U.S. and Canada at `1-800-98-BOXER (1-800-982-6937)` to order. Have your credit card ready; our sales representative will prompt you for the required information. From outside the U.S. and Canada call `+1-602-485-1635`. Business hours are Monday through Friday, 9 AM to 5 PM MST.

Ordering by Fax

Complete the form, and then click *Print*. Fax the order form to Boxer Software at `+1-602-485-1636`. Note that your credit card information will be encoded with a proprietary encoding algorithm for added security. We will decode the information after your order arrives. Our fax line is available 24 hours a day.

Ordering by Mail

Complete the form, and then click *Print*. Mail the order form to `Boxer Software, PO Box 14545, Scottsdale, AZ 85267-4545`. Note that your credit card information will be encoded with a proprietary encoding algorithm for added security. We will decode the information when your order arrives.

Ordering from Overseas

International Agents are available for those who might prefer to place their order with a local agent. Our agents accept payment in local currency and ship product from stock. Technical support services are also available.

Payment

Payment can be made in a variety of ways:

Credit Card
Visa, MasterCard, Discover or American Express

U.S. Check or Money Order
Made payable to 'Boxer Software'

Purchase Order
Purchase Orders can be mailed or faxed. Please make sure the Purchase Order includes both the shipping and invoicing addresses. Our payment terms are Net 30 days.

Western Union
Wire funds to 'David Hamel' and tell us the Control Number for the transaction, as well as the sender's name and the exact amount sent. Western Union also allows wire transfers to be made from their website: www.westernunion.com

U.S. Cash
Sent by certified or registered mail

PayPal
Send funds to 'sales@boxersoftware.com'. Don't have PayPal yet? Click here to sign up: www.paypal.com

International Money Order
Available at most banks. Money order should be drawn on a U.S. bank, in U.S. Funds, payable to 'Boxer Software'

International Postal Money Order
Available at the Post Office. Money order should be drawn in U.S. Funds, payable to 'Boxer Software'

Bank Wire Transfer
Please contact us for current bank transfer information. A $5.00 surcharge must be added to help offset the wire transfer fees we are assessed by our bank.
(**Note:** U.S. banks are not nearly as efficient as European banks with regard to bank wire transfers. Incoming transfers are slow, and receiving fees can be as high as $20.00. For this reason, we strongly encourage using another method of payment.)

8.2 Multi-User Licenses

A Multi-User License provides an inexpensive way for businesses, schools, universities or other work groups to supply their personnel with computer software in both a legal and cost efficient manner. By licensing Boxer for use on multiple computers you can standardize on a single editing tool that will serve the needs of all people within the group. In so doing, support and maintenance costs can be reduced, and users can benefit from having ready access to others who are using the same software. Multi-user licensing is also more economical than making individual purchases, because there is no need for us to supply extra disks, reference literature, etc. for all users within the group.

The organization purchasing the license designates a single individual to be the contact for shipping, technical support, upgrades, etc. We provide a single copy of the software package. You are then allowed to install Boxer onto as many CPUs as have been licensed.

The following chart details the cost of various Multi-User Licenses based on the number of CPUs. For example, if you purchase a license for 20 CPUs your cost would be

$425.40, which saves you 63.9% ($754.60) versus the cost of 20 individual purchases. The more copies licensed, the greater the percentage savings.

Note: Boxer Software's site licensing policy is based on 'individual use', not 'concurrent use.' When determining the number of licenses required, the computation should be made according to the number of CPUs on which Boxer will be installed or used, and not according to the maximum theoretical concurrent use that might occur. (An exception to this policy is made for individual, non-commercial users who may wish to install Boxer onto multiple CPUs which are owned by them, and of which they are the sole user.) Also, please note that the installation of Boxer onto a network server which is accessible by many users does not constitute use on a single CPU: the license quantity must be determined by the number of workstations that will use the software.

Number of Users	License Price	Discount Percentage
2	$100.30	15.0%
3	138.00	22.0
4	169.90	28.0
5	197.90	32.9
10	276.70	53.1
15	352.90	60.1
20	425.40	63.9
25	494.80	66.5
50	814.40	72.3
100	1383.60	76.5

If you later wish to add additional CPUs to the license, you can build upon the number of copies already licensed to achieve a better price. For example, the cost to grow a 20-CPU license to 25 CPUs is $69.40, which is the difference in cost between a 25-CPU and a 20-CPU license (494.80 - 425.40). In this way your license can be grown economically, as your needs change.

When it's time to upgrade to a new version of Boxer, the savings continue. The unit cost of a software upgrade will always be significantly reduced to our existing customers. Multi-User License upgrades are even further discounted by applying the original discount earned to the unit price of the upgrade, and then multiplying by the number of licensed CPUs. For example, if the unit price of an upgrade is $24.00, the price to upgrade a 10-CPU license would be: (100% - 53.1%) x $24 x 10 = $112.56.

The order form within Boxer will automatically compute Multi-User License pricing for quantities not shown in the chart above. That form can be found by selecting Order Boxer from the Help menu. If you have additional questions, or need pricing for 10,000 CPUs or more, please contact us.

8.3 Upgrade Information

Your purchase of Boxer entitles you to free bug fixes and minor enhancements as they become available, based on the version number at the time of your purchase. For example, if you first purchased version 14.0.x your base version is 14.0, and you are entitled to all updates which may be issued up to and including version 14.9.

When significant enhancements have been made to Boxer, the release level will become version 15.0.0, and the upgrade will be made available for a modest fee. Upgrade pricing will be established at the time of the upgrade, but in the past it has always been less than half of the original price of the software. Having purchased that upgrade, you would again become eligible for free upgrades up to version 15.9.

When upgrades are obtained by downloading from the Internet there is no shipping charge. When the customer asks that media be sent by postal mail, a shipping and handling charge will apply.

See also Ordering Boxer and Multi-User Licenses.


The Check for Latest Version command makes it easy to see if your version of Boxer is current.

8.4 Licensed User Benefits

There are many benefits that accompany the purchase of a fully licensed copy of Boxer.

Extended License
The evaluation version of Boxer can be used for up to 20 days to determine whether the software is suited to your needs. Please note that Boxer counts unique days of use--not simply elapsed calendar days--in an effort to provide a fair trial period. The purchase of a fully licensed copy of Boxer permits you to use the program indefinitely, in accordance with the software license.

Removal of Reminders
The evaluation version of Boxer uses various methods to remind the user of the need to order, and to encourage him to do so. The following reminders are removed in the fully licensed version:

- the popup dialog on program entry
- the message panel below the toolbar
- the watermark message at the bottom of printed pages
- the 'Trial Copy' message in the window title bar
- the 'Trial Copy' message in the minimized task button
- the dollar sign icon on the toolbar

Free Upgrades
Your purchase of Boxer entitles you to free bug fixes and minor enhancements as they become available, based on the version number at the time of your purchase. See Upgrade Information for full details.

Discounted Upgrades
For software upgrades which fall outside the version number range of free upgrades, Boxer Software will extend discounted pricing to its customers. Upgrade pricing will be established at the time of the upgrade, but in the past it has always been just a fraction of the original software price. Boxer Software will always extend preferred pricing to its customers on both new products, and on product upgrades.

Technical Support
Your purchase of Boxer entitles you to free technical support for one year from the date of purchase. See the Technical Support topic for information about the various ways in which support can be obtained.

8.5 International Agents

If you find it easier to do so, you may wish to order from one of our International Agents. Our agents accept payment in local currency and ship product from stock. Technical support services are also available.

The Netherlands, Belgium and Germany (and elsewhere in Europe)

Users in the Netherlands, Belgium, Germany and other European countries can buy Boxer licenses and upgrades from our local agent
since 1992, CopyCats S&S in the Netherlands, who also provide technical support during European office hours.

For current pricing and other information, please call or mail:

```
CopyCats Software & Services
P.O. Box 1088
1700BB Heerhugowaard
Netherlands

Phone: +31 (0)72 5745993
Fax  : +31 (0)72 5726559
Email: info@copycats.nl
```

9 Technical Support and Other Info

9.1 Technical Support

Menu: Help > Technical Support

Default Shortcut Key: none

There are several ways to receive technical support for Boxer. The first and most obvious resource is the online Help. Online help contains detailed information on the configuration and use of Boxer, and for all of its commands. If you are having problems, please consult the relevant section of help before contacting us for support. You may also be able to find answers to some common questions on our website:

www.boxersoftware.com

Email
You can send electronic mail to us via the Internet. We prefer this method of support since it allows us to fully research a problem before responding. Also, we can sometimes reuse an earlier reply for a problem which has been experienced by more than one person. We typically check email several times a day:

support@boxersoftware.com

Telephone
You can also reach us by telephone Monday through Friday, 10:00 AM to 4:00 PM, Mountain Standard Time.

```
Voice: +1-602-485-1635
```

Postal Mail or Fax
Finally, you can mail or fax your inquiry to us. If you choose one of these methods, please be sure to describe your problem fully and include any information which may help us to diagnose the problem. Whenever possible, please provide an email address so that we can make return contact quickly and easily.

```
Fax: +1-602-485-1636

   Boxer Software
    PO Box 14545
   Scottsdale, AZ
 85267-4545  U.S.A.
```

9.2 Software License - Evaluation Copies

Boxer Text Editor - Software License Agreement

COPYRIGHT:

Please read carefully the following terms and conditions. Installation and/or use of this product constitutes your acceptance of these terms and conditions, and your agreement to abide by them.

BY INSTALLING AND/OR USING THIS SOFTWARE YOU ACKNOWLEDGE THAT YOU HAVE READ THE LICENSE AGREEMENT, UNDERSTAND IT, AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. YOU ALSO AGREE THAT THIS AGREEMENT IS THE COMPLETE AND EXCLUSIVE STATEMENT OF AGREEMENT BETWEEN THE PARTIES AND THAT IT SUPERSEDES ALL PROPOSALS OR PRIOR AGREEMENTS, ORAL OR WRITTEN, AND ANY OTHER COMMUNICATIONS BETWEEN THE PARTIES RELATING TO THE SUBJECT MATTER OF THE SOFTWARE AND DOCUMENTATION.

GRANT OF LICENSE:
BOXER SOFTWARE GRANTS YOU, THE END USER, A NON-EXCLUSIVE PERSONAL LICENSE TO USE THIS SOFTWARE FOR A PERIOD OF UP TO 20 DAYS IN ORDER TO EVALUATE ITS SUITABILITY TO YOUR NEEDS. AFTER THE EVALUATION PERIOD YOU MUST EITHER PURCHASE A FULLY LICENSED COPY FROM BOXER SOFTWARE OR CEASE USING THE SOFTWARE. YOU MAY USE THE SOFTWARE ON A SINGLE PERSONAL COMPUTER SYSTEM AND MAKE AS MANY COPIES AS NEEDED FOR BACKUP AND ARCHIVAL. YOU MAY ALSO DISTRIBUTE THE SOFTWARE, UNMODIFIED AND IN ITS ENTIRETY, TO OTHERS WHO ARE INTERESTED IN EVALUATING THE SOFTWARE. YOU MAY NOT MODIFY, ALTER, TRANSLATE, DISASSEMBLE, DECOMPILE, RENT, OR LEASE THE SOFTWARE OR THE REFERENCE INFORMATION. THIS LICENSE IS EFFECTIVE UNTIL TERMINATED. YOU MAY TERMINATE IT AT ANY TIME BY DESTROYING THE SOFTWARE. IT WILL ALSO TERMINATE IF YOU FAIL TO COMPLY WITH ANY TERM OR CONDITION OF THIS AGREEMENT. YOU AGREE UPON SUCH TERMINATION TO DESTROY THE SOFTWARE.

WARRANTY:
TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, BOXER SOFTWARE AND ITS SUPPLIERS HEREBY DISCLAIM ALL WARRANTIES RELATING TO THIS SOFTWARE AND ITS DOCUMENTATION, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NON-INFRINGEMENT, WITH REGARD TO THE SOFTWARE, AND THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES.

LIMITATION OF LIABILITY:
IN NO EVENT SHALL BOXER SOFTWARE OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE PRODUCT OR THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES, EVEN IF BOXER SOFTWARE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN ANY CASE, BOXER SOFTWARE'S ENTIRE LIABILITY UNDER ANY PROVISION OF THIS LICENSE AGREEMENT SHALL BE LIMITED TO THE GREATER OF THE AMOUNT ACTUALLY PAID BY YOU FOR THE SOFTWARE PRODUCT OR U.S. $25.00. BECAUSE SOME STATES AND JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR

LIMITATION OF LIABILITY, THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

HIGH RISK ACTIVITIES:
The Software is not fault-tolerant and is not designed, manufactured or intended for use or resale as on-line control equipment in hazardous environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines, or weapons systems, in which the failure of the Software could lead directly to death, personal injury, or severe physical or environmental damage ("High Risk Activities"). Boxer Software and its suppliers specifically disclaim any express or implied warranty of fitness for High Risk Activities.

U.S. GOVERNMENT RESTRICTED RIGHTS:
The Software and documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraphs (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of the Commercial Computer Software-Restricted Rights at 48 CFR 52.227-19, as applicable. Manufacturer is Boxer Software, P.O. Box 14545, Scottsdale, AZ 85267-4545.

Unpublished-rights reserved under the copyright laws of the United States. Boxer Software, P.O. Box 14545, Scottsdale, AZ 85267-4545.

TRADEMARKS:
'Boxer' is a trademark of Boxer Software. Microsoft is a registered trademark of Microsoft Corporation. Other brand and product names are trademarks or registered trademarks of their respective holders.

9.3 Software License - Licensed Copies

Boxer Text Editor - Software License Agreement

COPYRIGHT:
The Boxer Text Editor, Boxer Help text and all supporting utilities are Copyright 1991-2010 by Boxer Software, All Rights Reserved Worldwide.

Please read carefully the following terms and conditions. Installation and/or use of this product constitutes your acceptance of these terms and conditions, and your agreement to abide by them.

BY INSTALLING AND/OR USING THIS SOFTWARE YOU ACKNOWLEDGE THAT YOU HAVE READ THE LICENSE AGREEMENT, UNDERSTAND IT, AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. YOU ALSO AGREE THAT THIS AGREEMENT IS THE COMPLETE AND EXCLUSIVE STATEMENT OF AGREEMENT BETWEEN THE PARTIES AND THAT IT SUPERSEDES ALL PROPOSALS OR PRIOR AGREEMENTS, ORAL OR WRITTEN, AND ANY OTHER COMMUNICATIONS BETWEEN THE PARTIES RELATING TO THE SUBJECT MATTER OF THE SOFTWARE AND DOCUMENTATION.

GRANT OF LICENSE:

BOXER SOFTWARE GRANTS YOU, THE ORIGINAL PURCHASER, A NON-EXCLUSIVE PERSONAL LICENSE TO USE THIS SOFTWARE UNDER THE TERMS STATED IN THIS AGREEMENT. YOU MAY USE THE SOFTWARE ON A SINGLE PERSONAL COMPUTER SYSTEM AND MAKE AS MANY COPIES AS NEEDED FOR BACKUP AND ARCHIVAL. YOU MAY ASSIGN YOUR RIGHTS UNDER THIS AGREEMENT TO A THIRD PARTY PROVIDED THE THIRD PARTY AGREES IN WRITING TO BE BOUND BY THE TERMS OF THIS AGREEMENT AND YOU TRANSFER ALL COPIES OF THE SOFTWARE TO THE THIRD PARTY, OR DESTROY ANY COPIES NOT TRANSFERRED. YOU MAY NOT COPY, MODIFY, ALTER, TRANSLATE, DISASSEMBLE, DECOMPILE, RENT, LEASE, OR ELECTRONICALLY TRANSFER THE SOFTWARE OR THE REFERENCE MANUAL. THE LICENSE IS EFFECTIVE UNTIL TERMINATED. YOU MAY TERMINATE IT AT ANY TIME BY DESTROYING THE SOFTWARE. IT WILL ALSO TERMINATE IF YOU FAIL TO COMPLY WITH ANY TERM OR CONDITION OF THIS AGREEMENT. YOU AGREE UPON SUCH TERMINATION TO DESTROY THE SOFTWARE.

LIMITED WARRANTY:
Boxer Software warrants that the Software, as updated and when properly used, will perform substantially in accordance with the accompanying documentation, and the Software media will be free from defects in materials and workmanship, for a period of ninety (90) days from the date of receipt. Any implied warranties on the Software are limited to ninety (90) days. Some states/jurisdictions do not allow limitations on duration of an implied warranty, so the above limitation may not apply to you.

Boxer Software's and its suppliers' entire liability and your exclusive remedy shall be, at Boxer Software's option, either (a) return of the price paid, or (b) repair or replacement of the Software that does not meet Boxer Software's Limited Warranty and which is returned to Boxer Software with a copy of your receipt. This Limited Warranty is void if failure of the Software has resulted from accident, abuse, or misapplication. Any replacement Software will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer. Outside the United States, neither of these remedies are available without proof of purchase from an authorized non-U.S. source.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, BOXER SOFTWARE AND ITS SUPPLIERS DISCLAIM ALL OTHER WARRANTIES AND CONDITIONS, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NON-INFRINGEMENT, WITH REGARD TO THE SOFTWARE, AND THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES. THIS LIMITED WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS. YOU MAY HAVE OTHERS, WHICH VARY FROM STATE/JURISDICTION TO STATE/JURISDICTION.

LIMITATION OF LIABILITY:
TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL BOXER SOFTWARE OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE PRODUCT OR THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES, EVEN IF BOXER SOFTWARE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN ANY CASE, BOXER SOFTWARE'S ENTIRE LIABILITY UNDER ANY PROVISION OF THIS LICENSE AGREEMENT

SHALL BE LIMITED TO THE GREATER OF THE AMOUNT ACTUALLY PAID BY YOU FOR THE SOFTWARE PRODUCT OR U.S. $25.00. BECAUSE SOME STATES AND JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY, THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

HIGH RISK ACTIVITIES:
The Software is not fault-tolerant and is not designed, manufactured or intended for use or resale as on-line control equipment in hazardous environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines, or weapons systems, in which the failure of the Software could lead directly to death, personal injury, or severe physical or environmental damage ("High Risk Activities"). Boxer Software and its suppliers specifically disclaim any express or implied warranty of fitness for High Risk Activities.

U.S. GOVERNMENT RESTRICTED RIGHTS:
The Software and documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraphs (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of the Commercial Computer Software-Restricted Rights at 48 CFR 52.227-19, as applicable. Manufacturer is Boxer Software, P.O. Box 14545, Scottsdale, AZ 85267-4545.

Unpublished-rights reserved under the copyright laws of the United States. Boxer Software, P.O. Box 14545, Scottsdale, AZ 85267-4545.

TRADEMARKS:
'Boxer' is a trademark of Boxer Software. Microsoft is a registered trademark of Microsoft Corporation. Other brand and product names are trademarks or registered trademarks of their respective holders.

9.4 Frequently Asked Questions

Menu: Help > FAQs

Default Shortcut Key: none

Frequently Asked Questions

After I order, will I get a password or key to convert the evaluation version of Boxer into a licensed version?
No. New software will be sent which is easily installed atop the evaluation version. All of your settings will be maintained. Software keys are frequently distributed on 'pirate' Internet sites, thus reducing sales and driving up the cost of software for paying customers.

Why can't other programs see the text I copied to the clipboard in Boxer?
You are almost certainly using an internal clipboard, rather than the Windows clipboard. Other programs can't see text that is placed on Boxer's internal clipboards. See the Set

Clipboard command for details.

Why am I having trouble opening filenames from Explorer when they contain embedded spaces?
This is due to a bug in Explorer. It doesn't enclose a filename in double quotes before sending it off to the associated application. In the file associations set up by Boxer's installer, double quotes have been added around "%1", so you'll find these associations (.TXT, .BAT, etc) work fine. But for any file associations you create yourself, or if you elected not to allow Boxer's installer to create the associations, you'll need to manually edit the association to have double quotes around "%1". You'll find that Boxer's help topic entitled 'File Associations' has additional useful information about this subject.

Why can't I see all my Windows fonts in the Screen Font dialog?
Boxer requires that *fixed width* fonts (monospace fonts) be used, so the Screen Font dialog box does not display *proportionally spaced* fonts. This is required, in part, to ensure that columnar selections can be highlighted neatly in rectangular blocks, and so that the Column Ruler can be used. These features would not be possible if the use of proportionally spaced fonts was permitted.

Will there be a German version of Boxer for Windows, as there was for earlier Boxer products?
It appears unlikely. We learned from our earlier products that the effort to release a program in a new language is quite substantial. It appears that our resources can be better spent enhancing our current products, or developing new ones.

Will there ever be a Linux version of Boxer?
That's uncertain at this time. We're keeping an eye on the Linux market, and will continue to do so.

How long did it take to develop Boxer for Windows?
The initial development took almost two years. Boxer for Windows was a ground-up effort, with almost none of the code from our earlier products being used in its development.

What language was Boxer written in? How many lines of code? What development tool was used?
Boxer currently consists of over 110,000 lines of C++ code. Borland's C++ Builder was used for development.

Where did the name 'Boxer' come from?

In the mid 1980's, one of the most popular editors for the PC was a product called BRIEF, which was then marketed by a company called UnderWare. In fact, the very first lines of Boxer/DOS were written using Brief, until Boxer was able to edit its own code. The name Boxer was simply a play on words: another style of men's underwear!

9.5 Credits

Special thanks are due to the following people for their patience, assistance and

friendship during the development of Boxer: Allen Day, Benjamin Whitehead, Glenn Rogerson, Mike Callahan, Roger Kimball, Mark Beiley, Daan van Rooijen, Ben Hamel, Adam LaChant, Bob Ellison, Carlos Tré and Dan Stratton.

Thank You!