EtCetera 1.20 Shareware Help Index

© 1992 Thetaware

EtCetera 1.20 is a shareware product. You are authorized to use it for a period of up to 45 days with no obligation. If you wish to use it beyond the trial period, you must register it. Refer to the banner screen for information on registering EtCetera.

EtCetera 1.20's help is set up differently than previous versions. Items are grouped based on function rather than by alphabetical order. Choose one of the following topics. If you are reading this help file for the first time, choose Introduction first.

Command Reference

EtCetera's commands are broken into functional groups.

Subject Index

To find out about :	Look at the following commands:
Controlling Program Flow	Break, For, Next
	Call
	End, ShutDown, Stop
	Goto
	If/Then
	Wait
Making Noise	Beep, PlaySound
Simulating Input	ChooseMenu
	KeyOff, KeyOn, Toggle
	Send, SendKeys, Type
User Input	Alert, Ask, Message, Query
	CommFileBox
	Input
	Wait
Window Manipulation	Activate, Close, Hide, Maximize, Minimize, Restore, Unhide, Window
	Align
	Bottom, Center, Left, Right, Top
	Flash
	Move, Size
	NextApp, PrevApp
Working with Clipboard	GetClipboard, SetClipboard
Working with Files	Call
	CD, ChDir, MD, MkDir, RD, RmDir
	Copy, Del, Delete, Ren, Rename
	Execute, Run
	FileFill, GetClipboard, ReadTo, SetClipboard, TitleFill, WriteFrom
	GetEntry, SetEntry
	Match

Alert, Ask, Message, Query

These functions are used to display messages or ask questions from the user. The format of these commands are:

Alert [string] {string} {options}
Ask [variable] [string] {string} {options}
Message [string] {string} {options}
Query [variable] [string] {string} {options}

Alert and Message display a dialog box which contains a single OK button, plus text provided by [string]. Alert also displays an exclamation point icon. If {string} is specified, it becomes the title bar text for the dialog box. This is the default if no {options} are specified.

If any options are specified, then Alert and Message behave exactly the same, and their default behaviors are provided only for backwards compatibility with prior versions of EtCetera. The options you can specify are as follows:

OK	- displays just an OK button
OKCANCEL	- displays an OK and a Cancel button
YESNO	- displays a Yes and a No button
YESNOCANCEL	- displays three buttons: Yes, No, and Cancel
RETRYCANCEL	- displays a Retry and a Cancel button
ABORTRETRYIGNORE	- displays three buttons: Abort, Retry, and Ignore
INFO	- displays an information icon ("i" in a circle)
EXCLAMATION	- displays an exclamation point icon
QUESTION	- displays a question mark icon
STOP	- displays a stop-sign icon
NOICON	- displays no such icon (the default)
DEF1	- the first button is the default button
DEF2	- the second button is the default button
DEF3	- the third button is the default button

For both Alert and Message, which cannot return the user's selection for your program, only the OK button option makes sense. You can, however, display the other button options - it just does not matter which button the user presses, and your program will not be able to determine it.

The Ask and Query commands are used to retrieve data. Ask, by default, has Yes and No buttons. Query has OK and Cancel. Both display a question mark icon. Selecting any options will override this default behavior. The [variable] can be any kind of variable. Based on which kind you choose, the following values will be placed into the variable based on the button pressed by the user:

Button	String	Numeric
OK	"OK"	1
Cancel	"Cancel"2	
Abort	"Abort"	3
Retry	"Retry"	4
Ignore	"Ignore"	5
Yes	"Yes"	6
No	"No"	7

String means a string variable or string array variable.   Numeric means any numeric variable - integer, integer array, long, or long array.

ChooseMenu

ChooseMenu can be used to select an item directly off of an application's menu bar without having to send keystrokes [see the description of SendKeys] to the application window.   Its format is

ChooseMenu [integer] [integer] {integer} ...

The first two [integer]s are required.   The first one indicates which menu in the menu bar to use (such as File, Edit, Search, View, etc.).   The second one indicates which item on the menu to choose.   If the item chosen is a cascading menu [refer to your Windows User's Guide], then an additional {integer} is required.   If a fourth level of menus is available, then another {integer} is required.   Etc.

Menus are zero-based, so the first item on a menu is numbered 0.   Also note that separators (lines which cross the menu) are items, so include them in your count.

For instance, if a menu bar contains two items, File and Edit, and the File menu has three items, Open, Save, and Exit, and the Edit menu has three items, Cut, Copy, and Paste, then to choose the File Save command, use

ChooseMenu 0, 1

and to choose the Edit Paste command, use

ChooseMenu 1, 2

CommFileBox

CommFileBox, new to 1.20, uses Microsoft's COMMDLG.DLL file to create a file list dialog box which is common among many of the Windows 3.10 "applets" (such as Notepad, Write, and Paintbrush). Thus, anyone who uses these programs will instantly recognize the intention of the dialog box.

The syntax of this command is:

```
CommFileBox [FILENAME [string variable] | PATHNAME [string variable]] [OPEN | SAVE] [string] {string}
```

CommFileBox can return either the simple filename of the user's selection, or the complete pathname, or both. To indicate which you want, use the keyword FILENAME or PATHNAME, followed by the variable into which you want the user's selection placed. You can use both FILENAME and PATHNAME - just be sure to follow each with a string variable.

You must specify either OPEN or SAVE to indicate which class of dialog box you want displayed. The SAVE keyword indicates that the dialog box should display the filenames in gray rather than in black, as with OPEN.

The required [string] is the title bar text of the dialog box. The optional {string} is the filename "filter" for the dialog box. The filter is a sequence of strings which appears in a drop-down list box ("combo box") and has associated with each string a certain file mask, such as *.TXT, *.EXE, *.PCX, and the like. Although it is usually of the form asterisk-dot-extension, it need not be.

The specify a filter, you must separate each item of the filter with a vertical bar character (|). For example, when you choose the File Open command in EtCetera, the combo box in the lower left corner of the dialog box displays

EtCetera Files (*.ETC)

and in the edit box in the upper left corner of the dialog box, you see

*.ETC

If you click on the down arrow of the combo box, you see

EtCetera Files (*.ETC)
All Files (*.*)

in the drop-down list box. If you click on All Files (*.*), then All Files (*.*) appears in the combo box, and *.* appears in the upper left edit box.

To set this up, the filter is setup as follows

```
EtCetera Files (*.ETC)|*.ETC|All Files (*.*)|*.*|
```

Note the vertical bar between each item. There are four items - two descriptions and two masks. Each mask must follow the description which describes it. There must also be a vertical bar at the end of the list.

So, for File Open, EtCetera itself uses something like:

```
CommFileBox Pathname $P, Filename $[100], Open, "EtCetera - Open File", "EtCetera Files (*.ETC)|*.ETC|All
Files (*.*)|*.*|"
```

Then $P is used to open the file, and $[100] is used to display the name of the file in EtCetera's title bar.

If/Then

The If/Then construction can be used to evaluate various circumstances while you EtCetera batch file executes. The general format is:

If [condition] {THEN} [command...]

The condition can be one of two things. The first is a comparison between two like items, such as strings or numeric values. To compare strings (which can be literal strings, string variables, or string array members), you use the following operators:

= or EC	tests for exact equality (case-sensitive)
<> or NC	tests for exact inequality (case-sensitive)
EI	tests for equality, case-insensitive
NI	tests for equality, case-insensitive

For example,

If $A = "Cancel" Then Goto EndOfProgram
If $A EC "Cancel" Goto EndOfProgram

These test to see if $A contains "Cancel", exactly, and, if so, transfers execution to the command following the label EndOfProgram. [See the description of the Goto command for more information.] The use of "Then" is optional.

To compare numeric values (which can be integers, longs, or either array type), use the following operators:

= or EQ	tests for equality
<> or NE	tests for inequality
< or LT	tests for "less than"
> or GT	tests for "greater than"
<= or LE	tests for "less than or equal to"
>= or GE	tests for "greater than or equal to"

For instance,

If ![12] <= 14 Then Message "![12] is less than 15"
If ![12] LE 14 Then Goto Failure

These both test ![12] to see if it is less than or equal to 14 (which is effectively the same as less than 15) and, if so, displays a Message dialog box or transfers program flow to the label Failure.

The second condition is used simply check something. With numbers, [condition] is true if the number is non-zero; [condition] is false if the number is 0. With strings, [condition] is true if the string contains anything; [condition] is false if the string is empty.

This can be used as follows:

If IsMin "Notepad" Then Maximize "Notepad"
IF $A THEN $[13] = UPPERCASE $A

See IsMin, Maximize, and UpperCase for a description.

Execute, Run

Execute and Run (which behave exactly the same) execute a program. The syntax is

```
Execute [string] {how}
Run [string] {how}
```

[string] is the command line used to run a program. This should be the name of the program file plus any required parameters, such as

```
Run "NOTEPAD WIN.INI"
```

This line would run Notepad and, because Notepad interprets the command line, Notepad would load the WIN.INI file. Note that the extension (.EXE) is not required.

The option {how} parameter is either MAXIMIZED (MAX), MINIMIZED (MIN), or WINDOWED (WIN), to tell EtCetera how to display the program initially. For instance,

```
Execute "NOTEPAD.EXE", MIN
```

would bring up Notepad as an icon. Keep in mind that running an application also makes it the active application, even if it is an icon.

Bottom, Center, Left, Right, Top

The commands move the active window (which must be a window, and neither an icon nor maximized) to the corresponding position on the display. Top moves the window up to the top of the display window (with no horizontal motion), and Bottom moves the window to the bottom of the display. Left and Right move the active window to the left and right sides of the display, with no vertical motion. Center centers the active window on the display.

None of these commands use any parameters.

FileFill, GetClipboard, ReadFrom, SetClipboard, TitleFill, WriteTo

These commands use the string array structure (variables accessed $[x] - see the description of strings for more details). TitleFill copies the title bar text of all open applications into the string array. GetClipboard reads all of the text data (assuming some text data is available) from the clipboard and places it into the string array. SetClipboard writes data from the string array to the clipboard, destroying whatever was there.

In each of these cases, the format of the command is

```
[command] [integer] {integer}
```

where [command] is the respective command, [integer] is a numeric value between 0 and 999 to indicate the beginning position in the string array to use, and {integer} is the last position to use in the string array. [Note that this behavior differs from previous versions.] If the last position is not specified, then it is assumed to be 999.

For example,

```
GetClipboard 201, 220
```

retrieves the first 20 lines of text from the clipboard and places it into string array items 201 through 220. If there are fewer than 20 lines of text in the clipboard, then the remaining string array entries are unchanged.

You can use the Lines keyword to get the number of string array entries actually changed by any of these commands.

ReadFrom, WriteTo, and FileFill are similar to the others, but they take an additional parameter:

```
[command] [string] [integer] {integer}
```

The [string] parameter is a filename for ReadFrom and WriteTo, and a file mask (including wild card characters * and/or ?).

ReadFrom reads lines from a file into the string array structure starting with entry [integer] and ending at entry {integer}. Again, if {integer} is not specified, it is assumed to be 999.

WriteTo reverses the process and writes data from the string array structure into a file. This functions creates the file if it does not exist, or destroys the previous file if it does exist, and then writes the lines to the file.

FileFill fills the string array structure with files which match the description given by [string]. For instance,

```
FileFill "*.EXE", 351, 400
```

fills the string array structure starting with position 351 with all files which end in the extension EXE, up to entry 400 (if 50 or more files in the current directory end in the extension EXE). The Lines keyword will return the number of string array items actually changed.

Move, Size

Size and Move change the position or size of the active window. If the active window is an icon or is maximized, then Size and Move do nothing.

Size [integer] [integer]
Move [integer] [integer]

The first [integer] is the horizontal position or width. The second integer is the vertical position or height. Both of these numbers are in terms of pixels, individual screen points. The upper left corner of the display is always coordinate (0, 0). The lower left corner of a standard VGA-type display, for example, is (639, 479), since a VGA display is made up of 640 pixels horizontally and 480 pixels vertically.

You can use the SCRW and SCRH keywords to get the width and height of the display window in pixels. Width and Height return the width and height of the active window. XPos and YPos return the horizontal (X) and vertical (Y) positions of the active window.

Activate, Close, Hide, Maximize, Minimize, Restore, Unhide, Window

These window functions are pretty self explanatory. Their format is

```
[command] {string}
```

{string} is the unique title bar text of the window on which the command is to operate. If {string} is omitted, it is assumed to be the active window. Note that

```
Activate
```

is a do-nothing command, which activates the active window. The active window is the one which receives user input, such as keystrokes or mouse actions. It is also generally, by default, the target of most of the things EtCetera does.

Close shuts down an application, closing its window(s). Hide makes a window disappear (but the application continues to run). Unhide restores a window so that it is no longer hidden. Maximize maximizes a window. Minimize changes the window to its iconic state. Window takes a maximized window or an icon and makes it a standard window. Restore takes an icon (only) and returns it to the state previously held immediately prior to becoming an icon - either a standard window, or a maximized window.

The unique title bar text is enough text of the title bar so as to uniquely identify the window. For instance, you can run Notepad more than once. Assume that in one copy (instance), you have loaded WIN.INI, and in the other, you have loaded AUTOEXEC.BAT. This means that the title bars of each instance of Notepad will be "Notepad - WIN.INI" and "Notepad - AUTOEXEC.BAT", respectively. To close the instance of Notepad with the WIN.INI, you need to use the command

```
Close "Notepad - W"
```

which is enough of the title bar text to uniquely identify which instance of Notepad you want to close. If you were to use the command

```
Close "Notepad"
```

there is no guarantee which instance will be closed (although certainly one instance will be closed).

You can, if you want, use the entirety of the title bar text:

```
Close "Notepad - WIN.INI"
```

But if you know you have only one instance open, it is much easier to use

```
Close "Note"
```

so long as no two windows have those same four characters at the beginning.

The title bar text is case-sensitive.

CD, ChDir, MD, MkDir, RD, RmDir

These directory-manipulation routines behave much like they do under DOS.   CD and ChDir change the active directory.   Note that, unlike DOS, you can (and must) use CD and ChDir to change the active drive.

MD and MkDir are used to create (make) a directory.   RD and RmDir destroy (remove) a directory.   Like DOS, the directory must be empty before it can be removed.

You can use literal text or any string with these commands.   For example,

```
CD C:\WINDOWS
CD "C:\WINDOWS"
```

are the same.   This allows you to use string variables to create, remove, and change to directories.

Copy, Del, Delete, Ren, Rename

These commands behave just like they do with DOS. Delete and Rename behave exactly like Del and Ren, respectively.

Copy copies files. Its format is

Copy [source] {destination}

[source] is the DOS filename, partial pathname, or complete pathname of the file(s) you want to copy. It can contain wildcard characters. {destination}, if specified, is where you want the files copied. If can be within the same directory (with a different filename), to a different directory, or to another drive. If {destination} is not specified, it is assumed to be the current directory with the same file names as the source. You can use either literal text or any string.

```
Copy C:\BACKUP\*.* D:\BACKUP
Copy "A:\*.DLL" SysDirectory
```

See the description of the SysDirectory keyword for more information.

Del and Delete delete files. The format is

```
Del [file mask]
Delete [file mask]
```

where [file mask] is a filename, partial pathname, or complete pathname, including possible wild card characters, indicating the file(s) you want to delete.

Ren and Rename can be used to rename files. Their format is

```
Ren [source] [new name]
Rename [source] [new name]
```

Examples:

```
Rename "*.BAT" "*.BAK"
Ren *.TXT "*.BKP"
```

Although [source] can be any valid DOS filename, partial pathname, or full pathname, including wild card characters, [new name] can only be a filename, including wild card characters, since rename changes the name of the file(s), but does not actually move it to another directory. Therefore, a pathname would be irrelevant.

The number of files affected by any of these commands can be determined by checking the value of the Files keyword.

End, ShutDown, Stop

The End command immediately stops execution of your program. Stop also stops execution of the program, but it automatically displays a dialog box with a stop-sign icon and an OK button. It uses the exact same syntax as the Message command, so you can replace the stop-sign icon. See the Ask, Alert, Message, Query topic for more information.

Neither of these commands are required to end a program - if there are no more lines of code to execute, EtCetera automatically stops. But you can use End to stop the program somewhere in the middle, if desired. Stop is generally for debugging purposes, allowing you to display program codes or other data while also ending the program. Using Stop is like using Message followed by End, but it takes only one command instead of two.

ShutDown also terminates an EtCetera program, but it also attempts to shut down Windows. If a DOS application is running, this will fail. If another program needs to save data, the other application will prompt the user to do so. If the user tells the other application to cancel the shutdown (which most application support), then Windows will not terminate - even though EtCetera will. Therefore, takes these situations into account if you want to use this command. The format is

Shutdown {integer}

where {integer} is an optional return code which can be retrieved with the DOS ERRORLEVEL keyword if you run Windows from a DOS batch file. Refer to your DOS documentation for more information.

NextApp, PrevApp

PrevApp and NextApp are used to activate a window indirectly. Windows, internally, keeps a list of windows, with the active window at the top of the list, the most recently active window as the next item on the list, and, at the bottom of the list, the window which has not been activated longer than any other.

NextApp activates the second window listed in the internal windows list (which was the active window prior to the active window being the active window). PrevApp activates the window at the bottom of the window list.

What this means is that NextApp can be used to ping-pong between two applications. When you use NextApp, the second window in the list becomes active, which places it at the top of the list, and the active window goes to position #2 on the list. Using NextApp again reverses the process.

PrevApp, on the other hand, can be used to round-robin through all open applications. This takes the window at the bottom of the list and places it on the top, effectively pushing all other windows down one position in the list. Doing this repeatedly will eventually activate every window.

Note that you cannot activate a hidden window.

GetEntry, SetEntry

These commands allow you to view and modify entries in initialization files, such as WIN.INI, SYSTEM.INI, or any other such file. Their formats are:

```
GetEntry [section] [entry] [string variable] {filename}
SetEntry [section] [entry] [string] {filename}
```

GetEntry retrieves an entry, and SetEntry sets it.

[section] is a string indicating the section name in the initialization file. A section is an area of the file with related entries, headed by text surrounded by square brackets [], such as [windows], [devices], or [386enh]. You do not use the square brackets when you indicate the section in GetEntry and SetEntry. [entry] is a string indicating the entry name. An entry name is followed by an equal sign, and the text following the equal sign is the value of the entry. [Do not include the equal sign when indicating the entry.] That value is what is specified by [string variable] and [string] above. GetEntry requires a string variable, since it has to have someplace to put the retrieved value. SetEntry can take any string (including an empty one) to change an entry.

These commands modify the file instantly and also make Windows aware of the fact of the change. This means that you can modify entries for programs on-the-fly, prior to running an application, without having to restart Windows (as you would have to do if you were to modify the WIN.INI file using Notepad). This will not, however, affect settings which Windows uses for initialization. For instance, you cannot use this to install drivers instantaneously by writing new entries to the SYSTEM.INI file. The data you write will be added, but it will have no effect on Windows until you restart it.

If you do not specify {filename}, it is assumed to be the WIN.INI file.

For example,

```
GetEntry "windows", "device", $D
SetEntry "extensions", "etc", "etcetera.exe ^.etc"
```

Send, SendKeys, Type

Send, SendKeys, and Type are synonyms of the same command. Each sends keystrokes to the active window. Note, however, that they do not behave as expected with DOS applications. In a future release, the Type command will be used to overcome this deficiency, so it is recommended that you use Send or SendKeys instead of Type.

The format of the command is

[command] [string]

where [command] is either Send, SendKeys, or Type, and [string] represents the keystrokes you want to send to the active window.

To send text, use

SendKeys "This is sample text."

To send any non-text keys, place the keytext in curly braces. For example, to press the Enter key, use

SendKeys "{Enter}"

Both text and non-text keystrokes can be included in the same command, so the above two examples could be combined as

SendKeys "This is sample text.{Enter}"

Function keys are also represented by including the key name in curly braces, such as

SendKeys "{F1}"

You can also send Alt, Ctrl, and Shift keystrokes. To indicate one of these keys, use /a, /c, or /s, respectively. For example, to send Ctrl+Alt+F4, use

SendKeys "/c/a{F4}"

To send a "/" character, use it twice:

SendKeys "//"

To send an actual curly brace, precede it with a slash:

SendKeys "/{"

Keypad keys can be sent by using "{KP_}", where the "_" represents the key. For example, the keypad version of "9" can be sent by using

SendKeys "{KP9}"

This also works for non-numeric keys: {KP+}, {KP-}, {KP*}, etc.

The typical 101-key keyboard has two copies of certain keys, called enhanced keys. These include the right Ctrl and Alt keys, the inverted T arrow keys (between the main keyboard and the keypad), and the six keys which make up the edit cluster directly above the inverted arrow keys. (The "unenhanced" version of most of these keys is available on the keypad when the NumLock key is off.)

You can specify these keys by using the "/e" prefix. Note that this prefix only "enhances" that which immediately follows it. So, since there are "enhanced" versions of the Ctrl and Alt keys, the following two commands result in different meanings:

```
SendKeys "/e/c{Del}"
SendKeys "/c/e{Del}"
```

The first means "hold down the right Ctrl key and press the Del key on the keypad". The second means "hold down the left Ctrl key and press the Del key on the edit cluster". Some programs specifically use the "same" key differently (standard versus enhanced), so this function allows you to access each individually.

Since you can apply "/e" to any key, you can create key combinations which do not exist.

Goto

The Goto command transfers the flow of execution from one location to another. The location must be specified with a label. A label begins with a colon, contains any text characters, and ends with a space.

For instance, take the following code fragment:

```
:Begin
Match $A, "*.EXE"
If $A = "" Then Goto NoFiles
Ask $B, "Found file: " & $A & ".  Use this file?"
If $B = "No" Then Goto Begin
Goto GotFile
:NoFiles
Message "No more files."
End
:GotFile
...
```

This program starts by checking the active directory for a file whose extension is EXE. If it does not locate one, then it transfers control to the line following the label :NoFiles. If it does find one, then it presents the user with its name in a dialog box, asking if the user wants to use the located file. If the user chooses the "No" button, then the program jumps back to the line after the label :Begin, to see if there are any other files with the EXE extension. If the user chooses "Yes" (the only other option for a default Ask command), then the program jumps to the line following the label :GotFile, which is illustrated here with "...". The program would at this point do whatever it needs to do with the file the user selected.

Labels are case-sensitive. Labels cannot be keywords, although they can be commands. In general, however, if a word is defined in EtCetera, it is recommended that it not be used as a label. For instance,

```
Goto End
```

means that there is a label :End somewhere. End is a command in EtCetera and, although you can use it, it is not recommended. However,

```
Goto Height
```

will fail. Height is a keyword, and EtCetera translates it.

Wait

Wait's purpose is pretty self-explanatory. It has a variety of uses.

If Wait is used with no parameters, EtCetera will display a dialog box, asking the user to click the OK button before continuing the program.

You can also follow Wait with an integer, such as

```
Wait 3
Wait 3 seconds
```

meaning that it should wait three seconds before continuing. You can also following it with the word "second" or "seconds" if you want. For that matter, you can follow it with the word "minutes" or "hours", but it always assumes seconds.

Another use is

```
Wait Closure
```

This form waits until the active window is closed (by the user). This allows you to permit the user to do something, like enter data into a dialog box in some application and then close it by clicking OK, before going on. The active window can be any window - an application or a dialog box.

The final alternative of this command is:

```
Wait Until [string] [what]
```

where [string] is the unique title bar text (see the description of the Activate command for details) of a window and [what] describes which event is desired. These can be either CLOSES, OPENS, MINIMIZES, MAXIMIZES, or WINDOWS. For instance

```
Wait Until "Notepad" Opens
```

causes EtCetera to wait until it can locate a window whose title bar text begins with "Notepad". If such a window already exists, then EtCetera continues right away.

Likewise, CLOSES waits until the first window it locates with the title bar text specified is closed (much like Wait Closure, except Closure is for the active window only). MINIMIZES, MAXIMIZES, and WINDOWS waits until the corresponding condition is met for the window in question. With CLOSES, WINDOWS, MINIMIZES, and MAXIMIZES, if no window with the specified title bar text does not exist, EtCetera continues execution right away. You can use the IsOpen keyword to check for the presence of a window prior to using any of these options.

Input

The Input command can be used to retrieve text data from the user.   Input displays a dialog box with an edit box and OK and Cancel buttons.   The user can enter text into the edit box and then press Enter (click OK), in which case the text the user enters will be placed into a specified string variable.   If the user clicks Cancel or presses Esc, then the string variable will be made blank.

The format is

```
Input [string variable] [string] {string} {string}
```

[string variable] is the variable into which the user's entry will be placed.   [string] is the descriptive text for the dialog box - a question, a request, or whatever.   The first {string} is the default value - this will be displayed in the edit box when the dialog box is first displayed.   The user can change this if desired.   The second {string} is the title bar text of the dialog box.   If you want to specify the title bar text, you must specify a default value.   If you want the edit box's default value to be blank, use "".

For example:

```
Input $[104] "Please enter your name:", "", "Logon"
```

Beep, PlaySound

PlaySound plays a wave file. A wave file is a digital recording of sound stored in disk in a file, usually with the extension WAV. In order to do this, you must have a Windows-compatible sound card and/or sound driver, and Windows 3.10 or higher, or Windows 3.00 with the Multimedia extensions. If you do not have a sound card, this command becomes a do-nothing command. Its format is

```
PlaySound {string} {options}
```

{string} represents a wave filename, partial pathname, or complete pathname. The options are SYNC, ASYNC, LOOP, and OFF. {string} is required for all options except OFF. SYNC tells EtCetera to wait until the wave file has completed playing before continuing. ASYNC tells EtCetera to start the file playing and then continue execution immediately. This feature works only if your sound hardware supports it.

The LOOP feature tells EtCetera to set the sound hardware to continuously play the sound file until explicitly told to stop. You can stop an ASYNC or LOOP wave file by using PlaySound a second time with a new file and setting, or use the OFF option, which terminates any sound in play.

Examples:

```
PlaySound "CHIMES.WAV", Loop
PlaySound "JFK3.WAVE", Async
PlaySound OFF
```

Beep beeps the system speaker, or plays the default beep sound, depending upon your Windows configuration. Beep takes an integer parameter, which indicates how many times to beep. With no parameter, it defaults to once.

Examples:

```
Beep
Beep 3
Beep 4 times
```

Break, For, Next

This series of commands is used to execute a piece of your program repeatedly. You set it up by setting a numeric variable to a start value with the For statement, then execute some code. Once the code you want to repeat is done, you use the Next command, which tells EtCetera to increment or decrement the variable by the amount you specified in the For command, then transfer control to the line following the original For command. This continues until the variable is either greater than or less then the stop value specified with the For statement, depending on whether or not EtCetera is incrementing or decrementing the value.

For's format is:

```
For [numeric variable] {=} [value] {TO} [value] {STEP} {value}
```

For example (pun intended):

```
For #A = 1 to 19 Step 2
Message $#A
Next
```

The first time through this, #A is equal to 1, and a dialog box appears displaying this value with an OK button. Once the button is pressed, the loop starts over (because of the Next command), and #A is equal to 3, since the Step value is 2. This continues until #A is greater than 19 or, in this case, when it equals 21.

You can reverse the values with

```
For #A = 19 to 1 Step -2
```

Note that if you want to decrement, you must use a negative step value or your loop will last quite a long time. If you do not specify a Step value, it is assumed to be 1 if the start value is less than the end value, or -1 if the start value is greater than the stop value. Also, the equal sign, the word "to", and the word "Step" are optional. You could also code the previous example:

```
For #A, 19, 1, -2
```

You may nest For/Next loops. What that means is that a For/Next loop may be wholly contained within another For/Next loop. For instance:

```
For #A, 1, 10
For ![3], 1, 10
#C = #A * ![3]
Message   "#A = " & $#A & ", ![3] = " & $![3], $#C
Next
Next
```

You will see 100 dialog boxes appear one after another, displaying the progression of #A and ![3], as well as their product in the title bar.

There is a limit to how deep you can nest For/Next loops. It can be no more than 10 deep. This does not mean that you are limited to 10 For/Next loops; what it means is that you cannot have more than 10 For statements without a corresponding Next statement. Each Next statement subtracts one from the number of available For's. The following illustrates the progression:

```
For                 1 For
```

For	2 For's
For	3 For's
Next	2 For's
For	3 For's
For	4
Next	3
For	4
For	5
For	6
Next	5
Next	4
Next	3
For	4
Next	3
Next	2
Next	1
Next	0

Don't ever write your code so that it exceeds 10 in the For count column.

The Break statement tells EtCetera to prematurely exit from the current For/Next loop. EtCetera will jump to the line following the next Next statement, forcing the termination of the current For/Next loop. This can be used as follows:

```
FileFill "*.ETC", 100
For #A = 100 to Lines + 99
Ask $A, "This one?: " & $[#A]
If $A = "Yes" Then Break
Next
If #A > Lines+99 Then Goto NoFileChosen
...
```

When a For/Next loop completes, the variable used in the For statement will be either greater than or less than the end value, depending on whether or not the end value is greater than or less than the start value, respectively. This can be checked at the end of a For loop to determine whether or not the loop was broken prematurely or if it completed.

See the descriptions of FileFill, Ask, If, and Goto for more information about the preceding example.

Match

Match is like a lesser FileFill command. It retrieves a single filename which matches the file mask provided. [By "mask" is meant a set of criteria - in this case, a filename or pathname, with or without wild card characters.] Match can be used repeatedly to retrieve additional files which match the file mask. The first time Match is called, it finds the first entry in the directory specified (or the current directory if no directory is specified) which matches the mask. Subsequent Match commands with the (exact) same mask locates the next entry which matches the mask. As soon as no additional matches are found (or if no match is found in the first place), a blank string is returned instead. Therefore, you can use this feature to find if a file exists by using the actual filename as the mask.

Once Match is used with a different mask, the process starts all over. So long as the same mask is used, Match will continue to retrieve files which match the mask (or until no more files match the mask). You cannot, therefore, use Match with two separate sets of masks simultaneously. You must use one mask first, and, once the desired file is located, move to the next mask - otherwise, you'll keep getting the same two files (the first ones to match each mask).

The format is

```
Match [string variable] [mask]
```

For example:

```
Match $A, "*.EXE"
```

Call

Call is used to load another EtCetera file and execute it from the beginning.   Its syntax is

```
Call [string]
```

where [string] is a filename or pathname to the EtCetera batch file you want to run.   EtCetera remembers all variables when a new file is called, but it does not remember the name of the previous file.   If you need to know the name of the calling file, save it in a variable.

Align

Align reorders all icons. If icons have been moved around the screen, Align repositions the icons neatly at the bottom of the display window. If the icons are already neatly arranged, this command does nothing. It takes no parameters.

Flash

Flash flashes the specified window a specified number of times.   If no window is specified, it uses the active window.   By "flash" is meant the title bar is toggled to the "activate" state and then "deactivate" state (or vice versa, if the active window is involved).   Usually, unless both states are specifically set to the same color, the title bar for an active window and an inactive window are colored differently.

The format is:

Flash {string} [integer] {time(s)}

{string} is the unique title bar text indicating which window you want to flash.   [See the description of Activate for information on creating unique title bar text.]   [integer] is the number of times you want to flash the window.   You can use a long variable for [integer], but it must not exceed the maximum value for an integer variable, 32767; It is very unlikely that you will want to flash a window more than 10 times, let alone 32767.

For example:

```
Flash 3
Flash "Notepad" 3 times
```

KeyOff, KeyOn, Toggle

The functions are used to set the states of the keys which have LEDs: NumLock, ScrollLock, and CapsLock. KeyOn turns on the LED; KeyOff turns off the LED; Toggle switches the state of the LED to its opposite. The format is

[command] [LED-key]

where [command] is the desired command, and [LED-key] is either a C, and N, or an S for CapsLock, NumLock, and ScrollLock, respectively. If desired, you can actually spell out the key name, but only the first letter is required.

You can only affect one key with a single command. If you want to turn on all three LEDs, use three separate KeyOn commands.

You can use the LEDs keyword to determine the state of these keys.

Special Keys

Below is a table which shows the special (non-text) keys supported by Send, SendKeys, and Type, and their representations. Note that the representation must be enclosed in curly braces (which must also be inside of quotation marks or in a string variable). This list does not include function keys or keypad keys, which are described in the description of Send, SendKeys, and Type. The representations are not case-sensitive.

Key	Representation(s)
BackSpace	Back, BackSpace, BkSp
CapsLock	Caps, CapsLock, CL
Del	Del, Delete
Down Arrow	Down, Down Arrow
End	End
Enter	Enter, Return
Esc	Esc, Escape
Home	Home
Ins	Ins, Insert
Insert	Ins, Insert
Left Arrow	Left, Left Arrow
NumLock	Num, NumLock, NL
Page Down	Next, PageDown, PgDn
Page Up	PageUp, PgUp, Prior
Pause	Pause
Print Screen	Print, Print Screen, PrtSc, SnapShot
Return	Enter, Return
Right Arrow	Right, Right Arrow
Scroll Lock	Scroll, Scroll Lock, Scrl, SL
Tab	Tab
Up Arrow	Up, Up Arrow

Keyword Reference

Keywords are words which are defined in EtCetera to have specific meanings.   Keywords behave like numeric values or strings, and can generally be used in place of them or within equations (but never in place of a variable where one is required).   The keyword descriptions are divided into groups based on the keywords' function.

Version keywords

DOSVer, ETCVer, WinVer

Version keywords return numeric values indicating version numbers. Each keyword returns the version number multiplied by 100, since EtCetera only supports integers (and cannot, therefore, use numbers with decimal points). There are three such keywords in EtCetera 1.20: ETCVer returns the version of EtCetera (always 120, since you are using 1.20); DOSVer returns the version of DOS (330 for DOS 3.30, 500 for DOS 5.00, etc.); WinVer returns the version of Windows (310 for Windows 3.10).

Time/Date keywords

Date, Day, Hour, Minutes, Month, Seconds, Time, Weekday, Year

There are nine time/date keywords in EtCetera.   Only two return strings; the rest are numeric.

The string keywords are Time and Date.   Time returns the time in military notation (i.e. 13:22:41).   Date returns the date in mm/dd/yy format (i.e. 10/24/92).

The numeric keywords are Day (returns the current date value), Month (the current month), Year (the current year), Hour (the current hour, in military 0-23 format), Minutes (the current minute value), Seconds (the current second value), and Weekday (the day of week).   Weekday returns a 0 for Sunday, a 1 for Monday, etc., through to 6 for Saturday.

Screen Coordinate keywords

Height, ScrH, ScrW, Width, XPos, YPos

Screen Coordinate keywords deal with the width, height, or position of items with respect to the video display. The video display is made up of individual points (called pixels). Different display types have different numbers of pixels both vertically and horizontally.

ScrW and ScrH return the width and height of the current video display in pixels, respectively. For a standard VGA-type display, this is 640 and 480.

Width and Height return the width and height of the active window in terms of pixels. A window which takes up one-fourth of the display area of a VGA display would have a width of 320 and a height of 240.

XPos and YPos return the x-coordinate and y-coordinate of the upper-left-most pixel of the active window. This is always with respect to the upper left corner of the video display, which is XPos = 0, YPos = 0 (written (0, 0)). If the upper left corner of an application's window is 30 pixels to the right of the upper left corner of the display and 70 pixels below it, XPos would return 30, and YPos would return 70. On a standard VGA display, the lower left corner of the display is (639, 479).

Window State keywords

IsMin, IsMax, IsOpen, IsWin

Window State keywords return values based on the current state of the active window. If a condition is true, these keywords return the value 1. If a condition is false, the return value is 0.

IsMax checks to see if the active window is in its maximized state. IsMin checks to see if the active window is an icon. IsWin checks to see if the active window is in its standard window state.

IsOpen checks to see if a window exists whose title bar text matches what you specify. For instance, you can check to see if Notepad is open and, if not, run it:

```
If IsOpen "Notepad" Then Goto :AlreadyOpen
Run "NOTEPAD"
:AlreadyOpen
...
```

See the descriptions of If/Then, Goto, and Run for more information.

Directory keywords

GetDirectory, SysDirectory, WinDirectory

Directory keywords return strings of various directories.   GetDirectory gets the current DOS directory, including a disk drive identifier.   For instance, if the current directory (i.e. if the program were "at the DOS prompt", what the prompt would show) is the root directory of your D drive, GetDirectory would return "D:\".

WinDirectory returns the complete pathname to the Windows directory, such as "C:\WINDOWS".   SysDirectory returns the complete pathname to the Windows system directory, which contains much of Windows itself.   This directory is almost always called SYSTEM, and it is always located in the Windows directory.   So, for the WinDirectory example, SysDirectory would return "C:\WINDOWS\SYSTEM".

String Conversion keywords

ANSI, Left, LowerCase, Mid, Right, UpperCase

String Conversion keywords allow you to modify and manipulate strings. UpperCase and LowerCase take a parameter (a string) and return it converted into either uppercase or lowercase, respectively. Left and Right take two parameters, a string and a numeric value, and return as many characters from the "edge" of the string specified as is specified. Mid takes three parameters, a string and two integers. The first integer indicates a position from the left "edge" of the string, and the second integer indicates the number of characters to return from the position indicated with the first integer. The ANSI takes an integer parameter and returns a single character whose ANSI value is the integer specified.

To illustrate this, assume that the variable $A contains "John Doe". In this illustration, $B will be used to store the modification.

$B = UpperCase $A

In this case, $B would contain "JOHN DOE" after EtCetera executes the line above.

$B = LowerCase $A

$B would contain "john doe".

$B = Left $A, 3

$B would contain "Joh"

$B = Right $A, 4

$B would contain " Doe"

$B = Mid $A, 3, 4

$B would contain "hn D"

The ANSI keyword is a little bit different:

$B = Ansi(169)

$B would be the copyright symbol, ©. Refer to your Windows documentation for a list of supported ANSI characters and their associated values. You can also use ANSI to represent a carriage return, which is ANSI value 13.

Note that in none of the above cases is $A changed at all - and, so, the name "String Conversion keywords" is a little misleading. In EtCetera, the only variable which is ever changed in an equation is the one to the left of the equal sign.

System State keywords

ActiveWindow, CPU, LEDs, Mode, NumApps

System state keywords return information about the behavior of the system. ActiveWindow returns the complete title bar text of the active window. CPU returns a string which indicates what kind of microprocessor is in the computer: "8086", "80186", "80286", "80386", "80486". "8086" could also be an 8088, and "80186" could be an 80188.

LEDs returns a three-character string which indicates the state of the LED keys on the keyboard. If the Caps Lock LED is on, the first character will be a "C". If the Num Lock LED is on, the second character will be an "N". If the Scroll Lock is on, the third character will be an "S". If any of these LEDs are off, its corresponding position will be a space instead of a letter.

Mode returns a string which indicates the current operating mode of Windows: "Real mode", "Standard mode", or "386 Enhanced mode".

NumApps returns the number of applications which are currently executing. It does not include EtCetera.

Other keywords

Clipboard, GetAttr, Length, Parameters, TempFileName

These keywords do not fall into any other category.

Clipboard returns the text in the clipboard. If there is more than 255 characters of text in the clipboard, EtCetera truncates the text.

GetAttr gets the file attributes of the file you specify. This attributes are defined according to the file's directory entry. Each attribute has a specific value (which is a power of 2, such as 1, 2, 4, 8, 16...).

Read-Only:	1
Hidden:	2
System:	4
Archive:	32

The following code illustrates how to show how to determine which attributes apply to a file:

```
#A = GetAttr "WIN.INI"
If #A >= 32 Then Message "Archive"
If #A >= 32 Then #A = #A - 32
If #A >= 4 Then Message "System"
If #A >= 4 Then #A = #A - 4
If #A >= 2 Then Message "Hidden"
If #A >= 2 Then #A = #A - 2
If #A >= 1 Then Message "Read-Only"
```

GetAttr takes a string for a parameter, which should be a valid, existing file.

Length takes a string for a parameter and returns the number of characters in the string. For example,

```
#L = Length "This is a test."
```

Parameters returns a string which is the command line parameters for EtCetera. For instance, if you run EtCetera with the command line (per the File Properties option in Program Manager)

```
ETCETERA GETFILES.ETC /COM1 /9600 /8N1
```

then Parameters returns "GETFILES.ETC /COM1 /9600 /8N1".

TempFileName retrieves a filename which is guaranteed to be unique within the system. The string returned is a complete pathname to a file which does not exist (already). This file will be in the directory indicated by the TEMP environment variable, which should be set in your AUTOEXEC.BAT file. Refer to your Windows documentation for more information. If the TEMP environment variable is not set, then the file will be in the Windows directory or the root directory of your boot drive.

You can use TempFileName to find a filename which you can use which will not accidently overwrite some other file. Do not use TempFileName if you do not plan to use the file right away, as Windows returns the filename, and it could just as easily give it someone else if the file is found to be nonexistent when another program requests a unique filename.

Data Formats and Variables

EtCetera 1.20 supports two data formats. The first is a numeric format. Numbers can be positive or negative, but they must be integers (i.e. they cannot have a decimal point). Do not use commas when writing numbers. To write ten thousand, use 10000 but not 10,000. Depending on the context, EtCetera can use numbers between negative two billion and positive two billion. [Technically, between -2147483648 and 2147483647. This is due to the way computers store numbers internally.]

The other data format is textual data, called strings (short for character strings, or strings of characters). A string can have a maximum length of 255 characters. Strings can also be empty.

Variables are storage locations in memory which can vary - hence the name. Variables are used to store data temporarily while a program operates on the data. EtCetera supports three kinds of variables. Two of these variable types are numeric. The third type is for strings.

Numeric variables

EtCetera supports two distinct numeric variable types. The first is a standard integer, and the second is a "long integer" or a long.

Integers may range from the value -32768 to the value 32767. [Again, this is due to the way computers store numbers internally.] If an equation or an assignment exceeds either of these values, the value will "roll-over". 32769, for example, becomes -32767. Long integers may range from -2147483648 to 2147483647.

An integer variable is specified with a pound symbol (#) and a single alphabetic character (A-Z). To assign a value to an integer variable, use an =, as

```
#A=32
```

Complex equations are supported, such as

```
#A = (12+#B) * #A
```

When EtCetera evaluates an equation, the order in which the equation is evaluated follows the standard mathematical evaluation criteria. Portions of an equation within parentheses are evaluated first. Then, multiplication (*), division (/), and modulus (%) are performed. (Modulus returns the remainder of a division operation.) Then, addition (+) and subtraction (-) are performed. Use parentheses to alter the order of operation. Parentheses may be embedded within parentheses.

In addition to the variables #A through #Z, EtCetera supports an integer array structure. This is a group of 1000 separate integers which can be accessed individually. To access one of these items, use

```
#[value]
```

where value is the entry number you want to use. Valid entry numbers are 0 through 999. For example,

```
#[100] = 13
#[#A] = 24
#[#[#A]] = 19
```

are all valid. The third entry shows a particularly powerful use of arrays. Entries can be nested. The innermost item is evaluated first, followed by that array item, followed by the outermost array item. Assuming #A is equal to

24, and #[24] is equal to 13, this command would set #[13] equal to 19.

Long variables are denoted with an exclamation point, as in !A or ![234]. Like integers, longs can be used in equations, and longs have their own array structure. You can mix and match integers with longs in equations, but if you assign a value to an integer which exceeds the limits of an integer variable, an error will occur.

Besides literal numerals and variables, there are keywords which return numeric values. These special words behave like numbers and can be used in place of them where a number (but not a numeric variable) is required.

String variables

A string is a series of alphanumeric and other characters. Strings are limited to 255 characters. A string variable is indicated with a dollar sign ($) followed by a single alphabetic character (A-Z). To assign a string to a string variable, use an =, as

```
$A = "Yes!"
$B = $A
```

Literal strings are designated by surrounding the text in quotation marks. To indicate a quotation mark should be a part of the text, place two quotation marks together: "Say ""what""" will be understood by EtCetera to be

Say "what"

EtCetera supports complex string functions or concatenation. Concatenation is performed by listing two or more strings, variables, or string keywords, and separating them with an ampersand (&). For example:

```
$A = "This "
$B = "is a "
$C = "test."
$D = $A & $B & $C
```

$D is "This is a test." after EtCetera executes these four lines.

If you want to convert an integer variable into text, preface the integer variable with a $, as in

```
Message $#C
```

which would display the value of #C, as text, in a dialog box. (See the description of the Message command for more information.)

In addition to the variables $A through $Z, EtCetera supports a string array structure. This is a group of 1000 separate strings which can be accessed individually. To access one of these items, use

```
$[value]
```

where value is the entry number you want to use. For example,

```
$[13] = "This is a test."
```

assigns "This is a test." to entry number 13. The valid numbers are 0 through 999. See the description of the integer array structure in Integers, above, for a more detailed description of the use of arrays.

The string array structure differs from the numeric array structure in that the string array structure plays an integral part (pun intended) of several of EtCetera's commands.   See the descriptions of FileFill, GetClipboard, ReadFrom, SetClipboard, TitleFill, WriteTo.

Introduction

EtCetera is a batch language interpreter for the Microsoft Windows environment. A batch language is a simple language which can be used to simplify or automate repetitive, redundant, or boring tasks. DOS has a batch language "built in", but Windows does not have one. [Please do not inform Microsoft of this oversight.] EtCetera adds this much-needed functionality to Windows. Using EtCetera, you can reduce a series of tasks to a double-click.

EtCetera is simple to program. EtCetera has an editor mode which allows you to enter commands into EtCetera and test them right away. To enter this mode, simply double-click the EtCetera icon in Program Manager. [If you use a different shell program for Windows, this may behave differently.] When you run EtCetera without any command line parameters, EtCetera enters editor mode.

If, instead, you specify a filename on the EtCetera command line [click once on the EtCetera Demo icon in Program Manager and then choose the Properties command from the File menu for an example], EtCetera loads the file you indicate and attempts to run the program. Once the program is done, EtCetera unloads itself from memory.

EtCetera batch files are simple text files. You can create them in editor mode, or you can type them in in Notepad. You can also create them in a word processor, but you must be certain to save them in text-only format.

Each line in the file is a single command and is terminated with a carriage return (press Enter). Only one command can be placed on a single line [see the description of the If/Then command for the only exception].

There are ten items of importance to EtCetera: Commands, Keywords, Mathematical Operators, Logical Operators, Modifiers, Variables, Numerals, Strings, Labels, and Comments. The paragraphs below describe these items briefly. The remainder of this help file more fully explains each.

Commands tell EtCetera to do something, such as run another program or simulate keystrokes. Commands frequently take parameters, which are other items which define the behavior of the command. This can be the name of the program to run or the keys to simulate.

Some commands have special words which are unique to the command (or group of commands), called Modifiers (or Options). The descriptions of each command list any Modifiers for a particular command.

Numerals are simply that - numbers - like 10000 or -18. In EtCetera 1.20, integer numerals are supported. You cannot use a decimal point with a number (a real number). Batch languages generally deal with concrete, whole things, not parts of things.

Strings (short for "character strings") are series of text. A string can contain up to 255 characters or as few as 0 characters (a blank string). Strings are surrounded by quotation marks.

Variables are storage areas in the computer's memory. Variables can be used to store numeric values or strings.

Keywords are specially reserved words which can be used in place of numerals or strings. Keywords behave as if they were the values they represent. For instance, the keyword DOSVer represents the version of DOS running on the system.

Mathematical Operators are used to create equations. These include +, -, and =. Strings can also be used in equations.

Logical Operators are used to make comparisons (and, at this point, are exclusive to the If/Then command). The equal sign = is a logical operator (and behaves differently than its mathematical twin when used this way). Other logical operators include < and >.

Labels are used to mark a specific line in a program. (See the Goto command.) A label is always preceded by a colon and consists of textual characters. It is terminated with a carriage return or a space. For example,

```
:ThisIsALabel
```

Comments can be used to describe sections of your program. A comment line is begun with a semi-colon, as

```
; This is a remark.
```

Everything on such a line is ignored by EtCetera. Using comments can make it easier to understand the behavior of your batch file, particularly if someone else is going to use it.

In general, you should separate commands, modifiers, equations, and some logical operators (those which are represented as text rather than symbols, such as < and =) with at least one space or comma. EtCetera uses these symbols to mark the end of one and the beginning of another. Spaces are not required within equations, so long as an operator of some sort falls between the various items in the equation. In general, adding spaces and commas to your programs will make it easier to read the program later. The sample code shown in the rest of this help file should help you understand the nuances of programming EtCetera. You can also review the sample code contained in the EtCetera Demo program, included with EtCetera, called ETC_DEMO.ETC.

Throughout the descriptions of commands and keywords, the formats of these commands show which parameters are required and which kinds of items are optional. Required items are surrounded by square brackets [], and optional items are surrounded by curly braces. In the sample code in this help file, a set of ellipses (...) are used to indicate that some other code would follow, but its presence is not required to illustrate the example.

Menu Commands

EtCetera will run in one of two modes. If EtCetera is run without including a batch file name on the command line, EtCetera will come up in editor mode. In editor mode, a window will appear in which you can type batch files and test them. This editor is similar to the Windows Notepad program. If you are familiar with standard Windows menus, then EtCetera's menus should be simple to use.

The File menu has six commands. The New command will erase whatever is in memory and let you start over from scratch. If something is currently in memory, you will be asked whether or not you want to save the file first. The Open command can be used to load a batch file from disk. Again, if something is currently in memory, you will be given the option to save whatever it is prior to loading a new batch file.

The Run command allows you to test batch files by beginning execution of the file. If you are done programming a file or need to go on to something else, you can use the Save and Save As commands to save whatever is in memory. The Save command will save the file using whatever the current file name is. This file name appears in the title bar at the top of the EtCetera window after the hyphen, like

EtCetera - AUTOMATE.ETC

This name is either the name of the file opened or the most recent name used to save the contents of memory. If, instead, the title bar reads

EtCetera - (untitled)

then no name has yet been assigned to this file, so EtCetera will prompt you for one.

If you want to save the contents of memory with a file name other than the one in the title bar, the Save As command will prompt you for a file name just as if the title bar read (untitled).

When you are finished using editor mode, the Exit command will close the EtCetera window and remove it from memory.

Under the Edit menu are four commands: Cut, Copy, Paste, and Goto. The Cut command will remove any highlighted text from the editor window and place it into the Windows clipboard. The Copy command will place a copy of any highlighted text from the editor window into the Windows clipboard. In order for these commands to be available, there must be some text highlighted. To highlight text using a mouse, move the mouse to the beginning position of the text you want to cut or copy, depress the left mouse button, drag the mouse in the direction of the end position of the text you want to cut or copy, and then release the left mouse button once the mouse pointer arrives at the end position of the text to cut or copy. To highlight text using the keyboard, move the caret (the blinking vertical bar) to the beginning position of the text you want to cut or copy using the arrow keys, then, hold down either shift key and use the arrow keys to move the caret to the end position of the text you want to cut or copy - then release the shift key.

The Paste command will take any text currently in the Windows clipboard and either insert it where the caret is located or, if some text is highlighted, will replace the highlighted text with the text in the clipboard, if available. If the clipboard does not contain any text data, the Paste command will be unavailable.

The Goto command will move the caret to the line whose number you specify. If an error occurs in your program, EtCetera will report the line number where the error occurred. You can use the Edit Goto command to quickly locate the offensive line.

The Insert menu has two items - Commands, and Keywords. Next to each of these is a small arrow, indicating that these are cascading menus - menu items which have menus. If you select one of these items, a second menu will appear. The Commands menu shows a list of all available commands in EtCetera. You can choose one of these, and EtCetera will insert the command with a description of required or optional parameters. The Keywords menu behaves similarly.

The Help menu has three items. The first is Index, which you use to access this help file. The second is Help On Help, which runs the Windows help program and loads a file which describes how to use help. If you have deleted this file, or if Windows cannot locate the file, then the Windows help program will display an error message.

The last item is the About EtCetera command. Choosing this item will display a dialog box with copyright information.