A Function Library For WINCMD

BY DOUGLAS BOLING

Our WINCMD utility gave you a command language interpreter for Windows 3.1. This issue enhances the WINCMD language by adding WCLIB, a DLL (dynamic link library) whose new functions will let you move and size windows, read and write files, access the Windows Clipboard, and even control your multimedia devices from a WINCMD program.

You can download both WINCMD.EXE and WLIB.WCL from the Utilities/Tips Forum of PC MagNet, as explained in the "Utilities by Modem" sidebar. You will also need to review the original WINCMD article (*PC Magazine*, April 27, 1993) to learn the keywords, syntax, functions, and so on that WCLIB augments. The C language source code and related files for WINCMD.ZIP and WLIB.ZIP are also available in the Utilities/Tips Forum of PC MagNet.

If you don't have a modem, you can still get the files without charge from *PC Magazine*. Send a postcard with your name, address, and preferred disk size (360K, 720K, 1.2M or 1.44M) to the attention of Katherine West, *PC Magazine*, One Park Ave., New York, NY 10016-5802. Or fax a request to 212-503-5799; no voice phone calls, please!

USING WCLIP To install WCLIB, you must copy WCLIB.WCL into the same directory as WINCMD.EXE. When WINCMD is started it will automatically add all the new WCLIB functions to the WINCMD language. You use the WCLIB functions exactly as you would the standard WINCMD functions: Just enter the name of the function, followed by its parameters enclosed in parentheses.

WCLIB's new functions can be divided into four functional groups: window management, file access, multimedia, and Clipboard. A table listing them all is shown in Figure 1.

WINDOW MANAGEMENT GROUP The function GetWindowHandle() returns the *handle* for a window on the screen. In Windows programming jargon, a handle is a number that a program uses when referring to a window. To get the handle of the Program Manager's main window, you would use the function

```
proghand = GetWindowHandle("Program
 Manager")
```

In WCLIB, knowing a window's handle is the key to controlling that window. The standard functions included in WINCMD, such as AppActivate(), always refer to windows by their title bar text. Unfortunately, not all windows have a title bar or other text. The WCLIB functions that involve windows use the window handle instead and so can reference any window on the screen.

With WCLIB, you can expand your WINCMD programs to give you complete control over your Windows desktop.

The next two functions, SizeWindow() and MoveWindow(), let a WINCMD program set the size and position of windows on the screen. SizeWindow() takes three parameters: the handle of the window, the new width, and the new height. The latter two parameters are specified in pixels. For example, to resize the Program Manager window to a window 300 pixels wide by 200 high, you'd use the following lines:

```
handle = GetWindowHandle("Program
 Manager")
SizeWindow (handle, 200, 300)
```

SizeWindow() returns TRUE (any nonzero value) if the function was successful and FALSE (a zero value) if it was not.

The MoveWindow() function similarly takes three arguments: the window handle, the horizonal position, and the vertical position. To parallel the previous example, to move the Program Manager window to the top-left corner of the screen, the commands would be

```
handle = GetWindowHandle("Program
 Manager")
MoveWindow (handle, 0, 0)
```

As with the SizeWindow() function, MoveWindow() returns TRUE if the function was successful in moving the window and FALSE if it was not.

While you often want to move and size a window, there are times when you simply wish to minimize or maximize it. To minimize a window to an icon, you use the function MinimizeWindow(), whose only parameter is the handle of the window. Similarly, MaximizeWindow() zooms a window to cover the entire desktop, and RestoreWindow() restores a window from an iconic or maximized state. Both also take only the window handle as a parameter. All three of these functions return TRUE if the function was successful and FALSE otherwise.

Since being able to learn the specific position and size of a window is just as important as being able to move or size it yourself, WCLIB includes three functions that return information in this area.

GetWindowSize() takes one parameter, the window handle, and it returns the size of the window. Both the width and height of the window are encoded in the one returned parameter. The width of the window is contained in the lower half of the returned value; the height is contained in the upper half. To separate the two values, you can use the HighWord() and LowWord() functions. For example, the following code fragment will display the size of the Program Manager's window:

```
size = GetWindowSize
 (GetWindowHandle("Program
 Manager"))
Say "The Program Manager window is"
 HighWord(size) "high and"
 LowWord(size) "wide"
```

As discussed in the last issue in connection with the WINCMD language, the SAY statement simply displays text in the WINCMD.EXE window.

Notice in this example that instead of using a separate line to get and save the handle of the Program Manager window, I simply included the GetWindowHandle() function in the call to GetWindowSize(). While this is not necessary, some people find it convenient to write programs this way. It reduces the number of lines in the program and eliminates the need to create a variable to hold the window handle. Note too that it might be good to check the GetWindowHandle() result before passing it on to GetWindowSize().

The GetWindowPos() function is similar to GetWindowSize(). Its single parameter is the window handle, and it returns the position of that window encoded in the returned value. Thus, by making only a few minor changes to the previous example, you can display the position of the Program Manager's window thus:

```
pos = GetWindowPos
 (GetWindowHandle("Program
 Manager"))
Say "The Program Manager window is
 at row" LowWord(pos) "and column"
 HighWord(pos)
```

GetWindowState() returns the state—iconic, restored, or maximized—of a window. This function takes a window handle as its one parameter and returns a 1 if the window is in a restored state, a 2 if the window is being displayed as an icon, and a 3 if it is maximized.

GetWindow() is a powerful function that can be used to track the relationships between a window and its parent, owner, or children. The function takes two parameters: the handle of the window, and a parameter that requests that a specified relationship will return a TRUE (nonzero) value. Figure 2 lists the relationships that can be requested and the value of the second parameter to request that relationship. GetWindow() returns the window handle for the window that has the specified relationship. If no window meets the relationship requested, or if either parameter passed to GetWindow() is invalid, the function returns FALSE.

Unfortunately, to explain these window relationships would take an entire article in itself. For those who are interested in learning more about the subject, the best text is the classic *Programming Windows,* by *PC Magazine* contributing editor Charles Petzold.

The GetWindowText() function is the inverse of GetWindowHandle(). It returns the title text of a window from the window handle passed to it. This function

The New WCLIB Functions

Window Management Functions	Action
GetWindowHandle (title text)	Returns the handle of the window with the matching title text
SizeWindow (window handle, cx, cy)	Resizes a window
MoveWindow (window handle, x, y)	Moves a window
MinimizeWindow (window handle)	Minimizes a window
MaximizeWindow (window handle)	Maximizes a window
RestoreWindow (window handle)	Restores a window
GetWindowSize (window handle)	Returns the size of a window
HighWord (number)	Returns the upper 16 bits of a number
LowWord (number)	Returns the lower 16 bits of a number
GetWindowPos (window handle)	Returns the *x* and *y* coordinates for a window
GetWindowState (window handle)	Returns the state (icon/restored/zoomed) of a window
GetWindow (window handle, relation)	Returns the handle of a window related to a window
GetWindowText (window handle)	Returns the title text for a window
PostMessage (window handle, message number, word param, long param)	Posts a message to a window
SendMessage (window handle, message number, word param, long param)	Sends a message to a window
LoadIconFile (icon filename)	Loads an icon and returns an icon handle
SetIcon (window handle, icon handle)	Sets a window's icon
Multimedia Functions	
SendMCIString (MCI string)	Sends an MCI command string
GetMCIErrorString (MCI error number)	Returns an error message for an MCI error number
File Functions	
FileOpen (filename, access mode)	Opens a file
FileMovePtr (file handle, offset, flag)	Moves a file read/write pointer
FileRead (file handle, bytes to read)	Reads bytes from a file
FileWrite (file handle, data to write)	Writes data to a file
FileClose (file handle)	Closes a file
FileExist (filename)	Determines whether a file exists
Clipboard Functions	
GetClipText ()	Retrieves text from the Clipboard
SetClipText (data for Clipboard)	Sets the Clipboard contents

Figure 1: **By adding the new functions provided by WCLIB.WCL, you can write far more powerful programs in the WINCMD command language. The functions are listed in the order they appear in the text.**

provides an easy way to convert a window handle returned by GetWindow() into a title that can be used by AppActivate() and the other functions of WINCMD that require title text.

The next two functions, PostMessage() and SendMessage(), give a WINCMD program access to the very heart of Windows: the message architecture. Messages are the commands and notifications sent to windows by other windows and by Windows itself. As with GetWindow(), a complete explanation of window messages is beyond the scope of this article, but the functions are included in the WCLIB repertoire to give WINCMD programs an additional level of functionality.

Both PostMessage() and SendMessage() take four parameters: the handle of the window to which the message is to be sent, the message value, the word parameter for that message, and the long parameter. PostMessage() returns a zero if the function fails, and a nonzero otherwise. SendMessage() returns the value returned by Windows' SendMessage() call.

A handy use for PostMessage() is to tell an application to terminate by posting a WM_QUIT message to it. For example, to tell the File Manager to terminate, you would use the following line:

```
PostMessage (GetWindowHandle("File
 Manager"), 0x12, 0, 0)
```

Windows has hundreds of different messages. Again, the best source for learning about these messages is Charles Petzold's *Programming Windows*. Since window messages are normally referred to by their names rather than by their actual number values, you will need to refer to the file WINDOWS.H, which is included with the Windows Software Development Kit and with other Windows development environments. You must be very careful when using SendMessage() and PostMessage(), since sending the wrong message at the wrong time can be disastrous to the target window and even to Windows itself.

The last two functions in the window management group are LoadIconFile() and SetIcon(). These are designed primarily to allow a WINCMD program to set the icon of its own window, but with care they can also be used to set the icons for other windows on the desktop. LoadIconFile() takes the filename of a Windows icon file as its only parameter. If the file is successfully loaded, LoadIconFile() returns a handle to the icon.

The two SetIcon() parameters are the handle of a window and the handle of the icon to use. The function returns the handle of the icon that the window used previously. For example, to set the icon of a WINCMD window with an icon from the file TRASH.ICO, the statements would be

```
hicon = LoadIconFile ("TRASH.ICO")
if (hicon <> 0)
  holdicon = SetIcon (hmain, hicon)
```

In this example, the file TRASH.ICO is loaded and the variable `hicon` is set to the icon handle. If the load fails, `hicon` will be zero. The IF statement allows the WINCMD window icon to be set only if the icon from TRASH.ICO was loaded successfully. Note that this example is the first use of WINCMD's predefined variable `hmain`, which is automatically set to the handle of the WINCMD window. The only other predefined variable in the WINCMD language is `hinst`, which is the instance handle for the current instance of WINCMD.EXE.

MULTIMEDIA FUNCTIONS Microsoft bundled its Multimedia Windows software with the standard Windows 3.1. The multimedia interface itself is fairly complex: It has over 100 APIs! Fortunately, however, Microsoft also provided a high-level interface, called the Media Control Interface (MCI), which provides a much simpler way to communicate with multimedia devices. Included in the MCI inter-

Utilities by Modem

The *PC Magazine* utilities are available by modem from PC MagNet, a ZiffNet service hosted by CompuServe.

To find the phone number nearest you, set your communications software to 300, 1,200, 2,400, or 9,600 bits per second, 7 data bits, even parity, 1 stop bit, and full duplex, then dial 800-346-3247 with your modem. When the modem connects, press Enter. At the HOST NAME prompt, enter PHONES. Follow the menus and note the number closest to you. Or you can call 800-635-6225 (voice) and follow the instructions and note the number.

To obtain the current issue's utility free of charge: Dial the local number; at the HOST NAME prompt, type CIS; and at the USER ID prompt, enter 60116,1. Then at the PASSWORD prompt, enter PCMAGUTIL.

To join ZiffNet: At the USER ID prompt, type 177000,5000. Then, at the PASSWORD prompt, enter PC*MAGNET. Finally, at the ENTER AGREEMENT NUMBER prompt, type PCMAG93.

Register your name and enter your American Express, MasterCard, or Visa number. (If you'd like to have your company billed instead, call CompuServe at 800-848-8990.) Your personal user ID and a temporary password will be displayed. A new password will arrive in the mail within ten days to confirm your subscription.

ZiffNet membership costs $2.50 per month. This includes access to *PC Magazine* Editors' Choice awards, Product Reviews Index, Weekly News from *PC Week*, Buyers' Market, ZiffNet Highlights, and the Support Forum (which also includes the current utility). CompuServe members can join by entering GO PCMAG at any CompuServe ! prompt.

Outside of these areas, PC MagNet costs $6.30 per hour for 300-bps service; $12.80 for 1,200 or 2,400 bps; and $22.80 for 9,600 bps. Billing is based on 1-minute increments.

GETWINDOW() Relationships

Value	Meaning
0	First sibling for a child window
1	Last sibling for a child window
2	Next sibling for a child window
3	Previous sibling for a child window
4	The owner of the window
5	The first child window of a window
6	The parent of a window

Figure 2: **The second parameter to GetWindow() must specify one of these allowed values.**

face is a call that allows programs to use ASCII strings to command multimedia devices. (Make sure you don't confuse these strings with those you may use to get your e-mail!)

The WCLIB SendMCIString() function allows WINCMD programs to send MCI command strings to a multimedia device. The single parameter to SendMCIString() is the command string to send. If the command sent was successful, SendMCIString() returns any response the command returns. If the command fails, SendMCIString() returns the string ERROR, followed by the MCI error number that was returned. For example, if you sent the command

```
SENDMCISTRING ("open cdaudio")
```

a response of ERROR 291 would indicate that the CD-AUDIO device was already in use.

The GetMCIErrorString() function lets a WINCMD program translate an error number returned by SendMCIString() into a string that describes the error. The one parameter required is the error number returned by SendMCIString(). Note, however, that a WINCMD program must parse the error string returned by SendMCIString() in order to separate the error number from its preceding ERROR string. For example, the following lines will display an error string if an MCI string command fails:

```
answer = SendMCIString (string)
if (Substr (answer,0 ,5)
 == "ERROR")
   say answer GetMCIErrorString
    (substr (answer, 6, 10))
else
   say answer
```

The MCIPLAY.WCM program shown later in this article further illustrates the use of these two multimedia functions.

FILE FUNCTIONS The WCLIB file functions allow you to read and write files. FileOpen() opens or creates a file for operation. The function takes two parameters: the name of the file to open and the access mode. The allowable access mode values are 0 for read-only, 1 for write-only, and 2 for read/write access. FileOpen() returns a file handle that must be used for all other file functions.

The FileMovePtr() function changes the position of the file read/write pointer; for example, the location in the file from which data will next be read or to which it will next be written. The function takes three parameters: the file handle, the pointer offset value, and a flag that indicates the method to be used when changing the pointer. The method flag can be set to one of three values. A 0 indicates that the file pointer should be moved to the specified offset from the start of the file. A method flag setting of 1 indicates that the offset should be made from the current file pointer position, and a 2 indicates that the pointer should be offset from the end of the file.

FileMovePtr() returns the new value of the file pointer for the file. For example, to set the file pointer to 100 bytes from the start of a file, the command would be

```
FileMovePtr (hFile, 100, 0)
```

To move the file pointer ahead 300 bytes,

WORDFIX.WCM

Complete Listing

```
//============================================================
// WinCmd program to fix problems with WinWord
//
// Copyright (c) 1993 Douglas Boling
//============================================================
//
// Check to see if Word already running.  If so,
// just switch to Word Window and exit
//
handle = checkrunning ("Microsoft Word")
if (handle) do
   appactivate (getwindowtext (handle))
   exit
end

//
// Launch Word and wait .5 seconds
//
"\winword.exe"

delay (500)

//
// Attempt to get handle to Word's main window.  If
// we can't quit.
//
handle = CheckRunning ("Microsoft Word")
if (handle == 0)
   exit

//
// Move and size window to my specs
//
MoveWindow (handle, 200, 15)
SizeWindow (handle, 900, 950)
exit

//------------------------------------------------------------
// CheckRunning - A routine that scans all windows to see
// if a window has a partial matching title text
//------------------------------------------------------------
checkrunning:
//
// Get first window in window list
//
handle = getwindow (hmain, 0)
//
// Loop until no more windows
//
while (handle <> 0) do
   //
   // Get window text
   //
   text = getwindowtext (handle)
   //
   // If the first part of the window text matches the
   // argument passed to CheckRunning return handle
   //
   if (substr (text, 0, length(arg(1))) == arg(1))
      return handle
   //
   // Get next window handle in window list
   //
   handle = getwindow (handle, 2)
end
//
// No matching window found, return 0
//
return 0
```

Figure 3: **The WINCMD WORDFIX program manages the way Word for Windows starts.**

PC Magazine Utilities Updates

As with all good software, the programs presented in *PC Magazine* are upgraded and improved. Here's a partial list of the programs on PC MagNet that have been updated. To download these files from the Utilities/Tips Forum, type GO ZNT:TIPS, type LIB or select Libraries from the menu, then select Library 2 PCMAG UTILS. Type DOW and the filename listed below (for example, DOW ANSI.COM), or select Download a File from the menu.

ADDIT.COM, Version 1.1
ANSI.COM, Version 1.3
BAT2EXEC.COM, Version 1.5
BCOPY, Version 1.2
CARDFILE.COM, Version 1.1
CHKFRAG.EXE, Version 1.7
CONFIG.CTL, Version 3.0
DIRMATCH.COM, Version 3.1
EMMA.COM, Version 2.2
EMS40.SYS, Version 1.1
PCACCESS.EXE, Version 1.1
RN.COM, Version 3.0
SLICE.COM, Version 1.3
SNIPPER.COM, Version 1.2
ZCOPY.COM, Version 1.4

For a list of the programs that are available on PC MagNet, download PCMCAT.ZIP from Library 1 (General Info) of the Utilities/Tips Forum.

you would use the command

```
FileMovePtr (hFile, 300, 1)
```

Similarly, to set the file pointer to 200 bytes from the end of a file use

```
FileMovePtr (hFile, -200, 2)
```

A common use for this last method of flag setting is to find the size of a file. If you move the file pointer to an offset of 0 from the end of the file, the returned value will be the file size. These examples all assume that the file has been previously opened and that the file handle has been assigned to the variable hFile.

The FileRead() function reads bytes from a file and takes as its two parameters the file handle and the number of bytes to be read. The return value for the function is the actual data read, which can be assigned to a WINCMD variable by using the equal sign (=), as explained in the last issue. The read is performed from the current position of the file pointer. After the read is done, the file pointer is updated to point to the byte following the last byte read. If there are no more bytes in the file to read—if the file pointer is pointing to the end of the file—FileRead() returns -1.

FileWrite() does the complementary job of writing to a file. It also takes two parameters, the handle of the file, and a *variable* that contains the data to be written. FileWrite() returns the number of bytes written to the file. If the file handle is invalid or if the file is read-only, FileWrite() returns 0.

The FileClose() function closes a file and takes as its only parameter the handle to that file. WINCMD programs should always close any files that have been opened before the program terminates.

Finally, the FileExist() function determines whether a specified file exists. The function takes the *name* of the file as its one parameter and returns TRUE if the file exists, FALSE otherwise. Note that the filename passed to FileExist() can contain wildcards.

CLIPBOARD FUNCTIONS The final two functions allow WINCMD programs to read and write text to the Windows Clipboard. GetClipText() takes no arguments and simply returns any text data in the Clipboard. If the Clipboard is empty or if the data there cannot be rendered in a text format (if it contains a bitmap, for example,) GetClipText() returns 0. SetClipText() empties the Clipboard and then sets it to whatever text has been passed to the function. The function takes one argument: the text to be placed in the Clipboard.

WCLIB AT WORK The WINCMD program, WORDFIX.WCM, shown in Figure 3, demonstrates some of the new functions of WCLIB in the process of solving two of my major problems with Word for Windows. One of these is that Word does not remember its size and location on the screen between launches. The other is that the application displays a message box demanding SHARE if a second copy is launched. WORDFIX fixes the message box problem by first checking to see if Word is already running and, if so, switching the original Word window to the foreground. If Word is not running, WORDFIX starts the program and then moves and resizes its window.

The WORDFIX subroutine CheckRunning checks the title text of every window on the desktop to see whether a Word window is currently open on the screen. This check consists of using GetWindow() to return the first window in the window manager list. (The window manager is the part of Windows that manages the windows on the desktop.) The routine then gets the title text for that window and compares it to the target window text—in this case, the string "Microsoft Word". If the text matches, CheckRunning returns the handle of the window. Otherwise, GetWindow is called again to get the next window in the window manager list and loops back. The loop ends after the last window has been checked.

The value returned by CheckRunning is compared with 0 to see whether Word is running. If so, AppActivate() is called to bring Word to the foreground. Since CheckRunning returns a window handle and AppActivate() needs the title text of a window, GetWindowText() is used to convert the handle into the necessary title bar text.

Note that the CheckRunning subroutine provides a way around the GetWindowHandle() limitation I mentioned toward the beginning of this article, namely, that the function will fail if a window adds text on its title line beyond what you have specified.

If Word is not running, WORDFIX launches it and waits half a second to allow Word to display its main window. CheckRunning is called again to return the handle to the Word window. If for some reason Word did not start, WORDFIX simply terminates. Otherwise, MoveWindow() and SizeWidow() are called to place the Word window exactly

where you want it on your screen. WORDFIX quietly terminates when it has completed its business.

One endearing feature of WINCMD programs is that they can be hidden behind any icon in the Program Manager. I have WORDFIX in my Programs group with the Word for Windows icon set instead of the default WINCMD icon. This way it looks as if I am simply starting Word, when in fact I'm using a WINCMD program to start the application.

The MCIPLAY.WCM program, which is shown in Figure 4, uses the MCI string interface to let you control multimedia devices on your system. The program is rather straightforward. It consists of the MCI string functions discussed earlier, the AskBox() function which is used to query the user for the MCI command, and a WHILE loop that continues the process until the user has pressed the CANCEL button on the dialog box. The SAY statement is used to display any return messages from the MCI.

If you have the file GONG.WAV in your Windows directory, you can hear MCIPLAY in operation by entering

```
open waveaudio
play waveaudio!\windows\gong.wav
close waveaudio
```

MCIPLAY.WCM

Complete Listing

```
//============================================================
// MCIPLAY - A WinCmd program that sends command strings using
// the SendMCIString function
// Copyright (c) 1993 Douglas Boling
//============================================================
//
// Ask the user for an MCI string command
//
string = AskBox ("Enter MCI String", " ")
//
// See if the user pressed the cancel button
//
if (a+string == a)
   string = EXIT
//
// Loop until cancel pressed
//
while (string <> exit) do
   //
   // Send the MCI string
   //
   answer = SendMCIString (string)
   //
   // Check for errors.  If error, print error string
   //
   if (substr (answer,0 ,5) == "ERROR")
      say answer getmcierrorstring (substr (answer, 6, 10))
   else
      say answer
   //
   // Get next command
   //
   string = AskBox ("Enter MCI String", string)
   if (a+string == a)
      string = EXIT
end
exit
```

Figure 4: **MCIPLAY is a WINCMD program for controlling multimedia devices.**

HOW WINCMD AND WCLIB WORK Space limitations in the earlier issue precluded a technical discussion of the WINCMD interpreter, so I'll review that program before considering WCLIB. The operation of the WINCMD interpreter centers around its variable list. This list contains all the keywords of the WINCMD language—IF, WHILE, DO, and so on—plus all the words that have been tokenized from the ASCII file that the interpreter is currently executing as a WINCMD program.

The process of running a WINCMD program can be divided into three steps: resetting the interpreter, loading and tokenizing the WINCMD program, and interpreting the program's token list. The first step involves loading the variable list with the WINCMD language keywords. Keywords are loaded into the list together with pointers to the routines in WINCMD.EXE that will actually execute the keywords' functions. Thus the IF keyword is loaded with a pointer to the LocalIF routine in WINCMD.EXE, for example. The command line parameters are then loaded into the variable list so that they will be accessible to the program. At this point, any function libraries with a .WCL extension are called to load their functions into the variable list as well. (I'll discuss what happens when a .WCL library is loaded shortly.)

Once the variable list has been initialized, the WINCMD program to be executed is read into a block of global memory. The file is then scanned, a word at a time, with the first character of each word determining how the WINCMD tokenizer treats the word.

If the word begins with a letter or an underscore, it is treated as a variable name. WINCMD.EXE then searches the variable list to see whether the word is already there. If it's not, the word is added to the variable list and a pointer for the entry is added to another list, called the *line token list.* (This list consists mainly of pointers to tokens in the variable list.) The order of the pointers in the token list is the same as the order of the words in the WINCMD program.

At this point the interpreter is converting the ASCII words in the WINCMD program into pointers to variables and keywords in the variable list. Thus when WINCMD is subsequently executed, the interpreter only has to work with the pointers and does not need to retokenize each line as it is executed.

There can be other items in the line token list besides pointers to entries in the variable list. If the word being parsed does not begin with a letter or an underscore but with a number, the interpreter converts the word into a 32-bit number that is added to the token list as a numeric constant. If the tokenizer sees a double quote character, it assumes that anything up to the next quote is a string constant. Instead of adding the quoted string to the variable list, the tokenizer adds it to the line token list as a string constant.

To summarize the tokenization process, let's look at the simple, three-line WINCMD program listed below.

```
FRED = 5
IF (BOB < 5)
 BOB = FRED + 3
```

This program begins as an ASCII file. After the file has been parsed, the variable list contains all the keywords plus two variable names, FRED and BOB:

```
While
Do
If
=
>
<
+
-
.
.
.
BOB
FRED
```

The original values assigned to the variables are the names of the variables, that is, FRED is assigned the string "FRED" and BOB is assigned the string "BOB". As shown in Figure 5, the token list contains such items as line-number tokens, variable-pointer tokens, and numeric-constant tokens. Note that the items in the line token list are in the same order as the words from the original file.

Once a WINCMD program has been tokenized, the only remaining task is to execute it, starting with the first item in the line token list. If the item is a pointer to a variable, the second item is checked to see if it is an equal sign. If so, the items beyond the equal sign are evaluated, and the result is assigned to the variable at the start of the line.

In the example shown in Figure 5, the first line starts with the variable FRED, so the interpreter checks to see if the second item is an equal sign. Since it is, the interpreter evaluates the rest of the items on that line and assigns the result (in this case 5) to the variable FRED. After the first line of the program has been executed, the entry in the variable list for FRED also contains the value 5. Since nothing has been assigned to the variable BOB, the data for that entry simply contains the string "BOB".

Line Token List

LINE 0	VAR (FRED)	OPERATOR (=)	CONSTANT (5)	
LINE 1	STATEMENT (IF)	VAR (BOB)	OPERATOR (<)	CONSTANT (5)
LINE 2	VAR (BOB)	OPERATOR (=)	VAR (FRED)	OPERATOR (+)	CONSTANT(3)

Figure 5: **The WINCMD interpreter uses a list of pointers like this one to execute its programs.**

The first item in the line token list for the second line is a keyword, so the interpreter calls the routine pointed to by the IF entry in the variable list. The IF routine then evaluates the items in the line token list that make up the condition on the line following the IF. Depending on the result of the evaluation, this line may or may not be executed. In the Figure 5 example, the variable BOB is still assigned the string "BOB", so its numeric value is 0. Since 0 is less than 5, the line following the IF statement is executed.

The third line of this program, like the first, is a simple assignment statement. The items beyond the equal sign are evaluated and in this case FRED is 5, so the result is 5 + 3 or 8. When the interpreter tries to execute the line after the last assignment, it sees the end-of-file token, meaning the program has finished.

Calling a predefined function is similar to processing a keyword. Function names are listed in the variable list, along with the keywords and variables. As with keywords, predefined functions each have a pointer in the variable list that points to the WINCMD.EXE subroutine that executes the function. Thus, executing a function is simply a matter of parsing its parameters and calling the routine that executes it.

ADDING LIBRARY FUNCTIONS Since functions are simply items in the variable list, it's fairly easy for libraries such as WCLIB to add functions. They are simply added to the variable list. When WINCMD.EXE is launched, it looks for files with a .WCL extension in its directory, and if any are found, LoadLibrary is called to load the file as a DLL. If the load is successful and the .WCL DLL has external entry points named WCLibLoad and WCLibFunc, it is assumed to be a WINCMD function library. The DLL WCLibLoad routine is called immediately so the library can perform any initialization processing necessary, and the WCLibFunc entry point is saved in WINCMD.EXE. Later, when a function from the library is called, the WCLibEntry point is called to execute the function.

When the interpreter is reset, a third entry point, WCLibReset, is called. The library then uses a callback function in WINCMD.EXE to add entries to the WINCMD variable list. These entries include the name of the function and numbers that indicate the source DLL of the function and a function number. The interpreter uses these numbers to determine which DLL to call when the function is called in a WINCMD program.

When the WCLibFunc routine in WCLIB is called, it uses the function number passed with the call to look up a pointer for the proper function to call inside the DLL. The WCLibFunc call also includes a pointer to a callback entry point in WINCMD.EXE. This allows libaries such as WCLIB to call routines that manipulate entries in the variable list. The callback is necessary because multiple instances of WINCMD.EXE may be running at any one time. The callback allows the single instance of the WCLIB DLL to operate on the correct variable list each time it is called.

Fortunately, the user need not appreciate the complex interaction between WINCMD.EXE and WCLIB. Together they provide a pówerful command language that can be used to control your Windows desktop. And the fact that WINCMD can be extended with additional libraries makes its language an ever-evolving process. Who knows? There may be another WINCMD function library sometime in the future! □

DOUGLAS BOLING IS A CONTRIBUTING EDITOR TO *PC MAGAZINE*.