Utilities

BY DOUGLAS BOLING

More Power for WINCMD

Last year, I wrote WINCMD, a batch-file language and interpreter for Microsoft Windows (for more information, see *PC Magazine,* April 27, 1993). WINCMD has become a popular addition to the library of *PC Magazine* utilities. Along with this popularity have come hundreds of requests for new functions and features. This issue's utility, WCL2 (for "WINCMD Library 2"), is a response to many of those requests.

WCL2 provides over 20 new functions for WINCMD. Among them are functions to read and write .INI files, parse filenames, and display the Windows common dialog boxes for File|Open and File|Save. The goal of WCL2 is to add functions that fill in the gaps left by WINCMD and the first function library, WCLIB (*PC Magazine,* May 25, 1993).

WCL2 requires a new version of WINCMD: WINCMD, Version 1.3. I designed WINCMD to be extensible without revisions to the basic program, but bugs in the original code necessitated this upgrade to WINCMD. To add a little sugar to the medicine of the WINCMD upgrade, I have added some new statements to the WINCMD language that should be popular among WINCMD fans. For details on these statements and other changes to WINCMD, read the accompanying sidebar "New and Improved: WINCMD 1.3 and WCLIB 1.1."

If you are interested in WCL2 and the new WINCMD, you may obtain them without charge from PC MagNet or by mail. Read the sidebar "Guide to Our Utilities & How to Get Them" for details. Both the executable files WCL2.WCL and WINCMD.EXE and their C-language source code files are available.

To install WCL2, just copy the file WCL2.WCL into the same directory where WINCMD.EXE is stored. Upgrading to Version 1.3 of WINCMD is equally simple: Just copy the new WINCMD .EXE over the old one. WINCMD 1.3 will run all your old .WCM programs. WCL2 does not replace WCLIB, my first WINCMD function library; both can be used together.

You can start WINCMD either by using the File | Run menu selection in Program Manager or by double-clicking on a WINCMD icon created in the Program Manager. When launched, WINCMD will load the WCL2 library (and WCLIB, if present) automatically. You can confirm that WCL2 is loaded by selecting the Help menu in the WINCMD window; if WCL2 is loaded, you'll see an About WCL2 menu item in the Help menu. The new functions of WCL2 also will be listed in the WINCMD Help dialog.

THE NEW FUNCTIONS The functions added by WCL2 are shown in Figure 1. The first group of functions is string functions, designed to make working with text strings easier. The Val and Chr functions work like the Val and Chr$ functions in BASIC. Val returns the number associated with the ASCII character passed to the function. For example, Val(A) returns the number 65, which is the ASCII code for the letter "A". If you pass a multicharacter string, the first character will be used.

The Chr function is just the opposite of Val; it converts the passed number to its associated ASCII character. For example, Chr(72) returns the letter "H". Val and Chr are handy when you need to write your own string processing functions. When you use these two functions, any character can be converted to a number, processed, and converted back to a character.

The Reverse function reverses the order of the characters in a string. For example, Reverse("abcdefghi") would return the string "ihgfedcba". Since WINCMD does not differentiate between numbers and text strings, Reverse can also be used to reverse the order of digits in a number. For example, the call Reverse(12345) will return 54321. Using Reverse in combination with the WINCMD functions SubStr and InStr, you can process strings from back to front instead of front to back.

The InStr function searches a string for the first occurrence of a substring. If the substring is found in the string, InStr returns the zero-based index of the start of the substring in the original string. For example, InStr ("qwerty", "er") will return the value 2. If the substring is not found, InStr returns -1.

InStr is not case-sensitive, so the call InStr ("abcdef",DEF) will find the string "DEF" in "abcdef" and return the number 3. InStr is handy for parsing long strings, especially when it's used with SUBSTR.

The FullPath function returns a fully specified path and filename from a relative filename. If you pass FullPath a filename such as WINCMD.EXE, it attaches the current drive letter and path to the name and returns C:\PCMAG\UTIL \WINCMD.EXE. The filename passed to FullPath can use partial paths, as in

Top 10 Utilities

Rank	Utility
1	**PRN2FL** - *Captures printer output*
2	**MONTOR** - *Monitors system resources*
3	**PCDEZP** - *Extracts .ZIP files*
4	**INPUTT** - *Text-input batch files*
5	**WTIME** - *Keeps PC clock accurate*
6	**BAT2EX** - *Compiles batch files*
7	**CONFIX** - *Issues boot-time commands*
8	**FONED1** - *Creates mnemonic phone numbers*
9	**TED** - *Tiny screen editor*
10	**PCMCVT** - *Strips word processing codes*

This list is based on the total number of downloads each utility has had on PC MagNet over the past 12 months. To download these files, type GO ZNT:TIPS. For more information, see the sidebar "Guide to Our Utilities & How to Get Them."

WCL2's New WINCMD Functions

String functions	
Val (*character*)	Returns the ASCII number for a character
Chr (*number*)	Returns the ASCII character for a number
InStr (*string, substring*)	Returns the index in string of substring
Reverse (*string*)	Returns the string in reverse order
Filename functions	
FullPath (*filename*)	Returns the full path of a file
GetFileDrive (*filename*)	Returns the drive of a file
GetFileDir (*filename*)	Returns the directory of a file
GetFilePath (*filename*)	Returns the drive and directory of a file
GetFileName (*filename*)	Returns the name of a file
GetFileExt (*filename*)	Returns the extension of a file
File-processing functions	
FileOpenBox (*filter name, filter filespec*)	Displays a File Open dialog
FileSaveBox (*filter name, filter filespec*)	Displays a File Save dialog
Keyboard-monitoring functions	
IsKeyDown (*sendkey key*)	Returns nonzero if key is being pressed
IsKeyShifted (*sendkey key*)	Returns nonzero if key is shifted
Window functions	
SetWindowText (*window handle, text*)	Sets the title text of a window
ShowWindow (*window handle, show param*)	Hides a window or makes it visible values for *show param*: WINDOW_SHOW Show window WINDOW_HIDE Hide window
IsRunning (*program name*)	Returns nonzero if program is currently running
Date and time functions	
GetDate	Returns the system date (mm/dd/yy)
GetTime	Returns the system time (hh:mm:ss)
.INI file functions	
ReadINIData (*section, key, default value, .INI filename*)	Returns data from an .INI file
WriteINIData (*section, key, default value, .INI filename*)	Writes data to an .INI file

***Figure 1:* The second WINCMD add-on library adds functions in a variety of categories.**

..\TEMP\WINCMD.EXE. In this case, FullPath gets the current directory, backs up one directory because of the .., and then adds the TEMP directory to the path. If the current drive is F: and the directory is \PROJECTS\WINCMD, FullPath would return F:\PROJECTS\TEMP\WINCMD.EXE.

FullPath is handy when you need to parse a filename entered by a user. With FullPath, WINCMD programs can parse filenames without having to worry about relative directories or the lack of a drive letter.

Note that the path and filename a user enters will be returned in a fully specified format even if the filename doesn't exist. The purpose of FullPath is to fill out a file's path information, not to verify the file's existence. To determine if a file and path exist, use the WINCMD function FileExist, which is part of WCLIB.

The GetFileDrive, GetFileDir, GetFileName, and GetFileExt functions are designed to ease the burden of parsing filenames. GetFileDrive returns the drive on which a file is located. The filename passed to GetFileDrive or any of the related functions does not have to be specified in its entirety. If only a filename without a drive is passed, GetFileDrive will return the current default drive. If a drive is specified in a filename, that drive letter is returned. The functions GetFileDir, GetFileName, and GetFileExt return the path, name, and extension of the passed filename. As with GetFileDrive, these other functions do not need a fully specified path. The last member of this family, GetFilePath, returns both the drive and the directory of a file.

Since many users write WINCMD programs that process files, WCL2 includes two functions that make asking for those filenames more user-friendly. The FileOpenBox function displays Windows' File Open dialog box used by most Windows programs, including WINCMD.

One feature of the File Open dialog box is a filter that allows the dialog to display only those files with a particular extension. Users can change this by selecting from the combo box in the lower-left-hand corner of the dialog. When using the FileOpenBox function, a WINCMD program can specify the filter to be used by passing two strings: a description of the filter and the actual DOS file filter with wildcards.

For example, to ask the user to select from bitmap files that have the extension .BMP, the call would be

```
FileOpenBox ("Bitmap Files", "*.BMP")
```

When the File Open dialog box is displayed, the filter *.BMP will be in the filename selection, and only files with a .BMP extension will be displayed in the file list.

You can combine multiple DOS filters by using a semicolon as a separator. For example, to list run-length-encoded bitmaps with the file extension .RLE along with .BMP files, the call would be

```
FileOpenBox ("Bitmap Files",
 "*.BMP;*.RLE")
```

The FileOpenBox function also adds the selection "All Files" to the filter list. If the user selects a file, FileOpenBox returns the filename; if the user clicks the Cancel button, FileOpenBox returns -1.

The FileOpenBox function does not check to see whether the file exists before returning to the WINCMD program. To warn a user that a file does not exist, you could use the following code fragment:

```
NeedFileName = 1
while (NeedFileName) do
   Name = FileOpenBox ("Text Files",
   "*.txt")
   if (Name = -1)
      NeedFileName = 0
   else do
      if (NOT FileExist (Name)) do
        rc = MsgBox ("The file "
        +name+" does not exist!
        Continue?", "WINCMD",
        MB_YESNO);
      if (rc = ANS_YES)
         NeedFileName = 0
   end
end
```

The FileSaveBox function displays Windows' File Save dialog box. FileSaveBox allows a filename filter to be specified in the same manner as in FileOpenBox. The function returns either the filename selected by the user or -1 if the Cancel button was clicked.

The next two functions, IsKeyDown and IsKeyShifted, assist WINCMD programs in monitoring the keyboard. The IsKeyDown function returns a nonzero value if the user is currently pressing the specified key. The function takes one pa-

What's New in WINCMD 1.3 And WCLIB 1.1

The new version of WINCMD, available now on PC MagNet, is much improved over the original WINCMD. This version not only has some new and quite useful statements, but other areas of the program have been improved as well. Readers using an earlier version of WINCMD will definitely want to download WINCMD, Version 1.3.

Improvements to WINCMD started with WINCMD 1.1. In Version 1.3, I added a help screen that lists the syntax of the WINCMD language, as well as all the functions available to WINCMD programs. This help screen automatically queries WINCMD function libraries such as WCLIB and WCL2 for their functions as well, so the help screen is continually up to date. In WINCMD 1.1, I also added a GETENVVAR function, which returns the value of environment variables.

Version 1.2 of WINCMD was a maintenance release that corrected a number of nagging problems that WINCMD users were seeing. No new functions were added in 1.2.

The most dramatic changes are in Version 1.3, which adds three new statements (OnDrop, LaunchSpecial, and ClearOutput) and two new functions (GetDropCnt and GetDropFile).

LaunchSpecial, the most requested improvement to WINCMD, allows a WINCMD program to control how a launched program is displayed and can force WINCMD to wait until the launched program completes before continuing on to the next statement in the program. One warning: LaunchSpecial will not work if your _DEFAULT .PIF does not have Background Execution checked.

The syntax of LaunchSpecial is simple; just use the statement LaunchSpecial followed by any launch conditions and then the command you would normally place in a WINCMD program. For example, in order to start Microsoft Word for Windows and wait until it completes before continuing, the statement would look like

```
LaunchSpecial (LC_WAIT) WINWORD
```

The parameter LC_WAIT can be combined with the launch parameters LC_MAXIMIZE, LC_MINIMIZE, or LC_BACKGROUND using the bitwise OR operator ORB.

This statement has a number of valuable uses, some of which are demonstrated in the example program, TrashCan. When used with the LC_WAIT parameter, LaunchSpecial can return the exit code the child program returns to DOS and Windows when it completes. Many programs use exit codes, which can range from 0 to 255, to return valuable information about the success or failure of the program. WINCMD programs can access this return value by checking the predefined variable RETURNCODE. WINCMD sets RETURNCODE with the return-code value of the last program executed with LaunchSpecial.

It is possible to force WINCMD to continue to the next statement even if the program being waited for is still running. Just click on WINCMD's Run button, and your WINCMD program will continue executing. If you force WINCMD to stop waiting, the value of RETURNCODE will be greater than 65,535—an impossible value if the program had actually completed.

The OnDrop statement allows WINCMD programs to take advantage of a drag-and-drop interface. The syntax of OnDrop is

```
OnDrop {statement}
```

where *statement* is any WINCMD statement. Normally the statement will be a call to a user-defined WINCMD function, but it can also be any other valid WINCMD statement, including a DO-END block.

When WINCMD executes an OnDrop statement, the statement on the OnDrop line is not immediately executed. The program continues to the statement after OnDrop, and Windows is informed that the WINCMD window can accept dropped files. When a user drops a file on the WINCMD window, WINCMD finishes executing the line it is currently on, and then executes the statement on the OnDrop line. When the OnDrop statement has completed, WINCMD returns to the next line beyond the last one it had executed before the files were dropped. If the statement on the OnDrop line is a call to a user-defined function, the complete function is executed when a file is dropped. To disable the OnDrop statement, just use OnDrop without any trailing statement. WINCMD will inform Windows that it no longer accepts drag-and-drop files.

Two functions assist OnDrop in processing the dropped files. GetDropCnt returns the number of files or directories that were dropped on the WINCMD window at any one time. Note that this is value is not cumulative; if five files are dropped, then one file is dropped, GetDropCnt will return a value of 1. GetDropCnt takes no parameters.

GetDropFile returns the name of a file dropped on the window. GetDropFile takes one parameter: the zero-based index into the number of files dropped. For example, if three files were dropped, GetDropFile(0) would return the first file dropped and GetDropFile(2) would return the third file dropped.

The simple example of OnDrop

shown below prints a list of the files dropped on the WINCMD window:

```
// Tell Windows we will accept
// dropped files
OnDrop DisplayFiles()
// Wait for 1 minute for files to
// be dropped
Delay (60000)
// Unregister OnDrop function
OnDrop
//Terminate the program
exit
----------------------------
// This routine will be called
// when files are dropped.
-----------------------------
DisplayFiles:
   count = GetDropCount()
   while (count > 0) do
      count = count - 1
      say GetDropFile (count)
   end
return
```

ClearOutput is the last and simplest of the new statements. ClearOutput clears the WINCMD output window used by the PRINT and SAY functions. The idea is that some users may want to use the output window to display data, so ClearOutput provides an easy way to reinitialize the WINCMD output window.

There have also been improvements to WCLIB. Version 1.1 adds error checking of file operations and some new constants that should help with the FileOpen command, which can easily fail depending on the state of the file. In the past, WINCMD programmers had to remember numeric values to tell the FileOpen whether a file should be opened in read-only or read/write mode and whether a file should be created if it didn't exist before. Now these constants are automatically available in the FILE_READONLY, FILE_ READWRITE, and FILE_CREATE variables. The FILE_CREATE constant can be combined with the FILE_READONLY or FILE_ READWRITE constants using the ORB operator. □

rameter: a character in the WINCMD SendKeys format, which is a string of characters that specifies a key.

For the most part, a character in SendKeys format is simply the character in quotation marks. For example, to get the state of the A key, the call would be IsKeyDown("A"). For some keys, such as the Tab key or the Backspace key, however, a simple character will not do. For these keys, a string is passed to the function enclosed in curly brackets. To determine if the Tab key is down, for example, the call would be IsKeyDown ("{tab}"). A complete list of tags for SendKeys characters can be found in the Help window of WINCMD.

You can combine calls to IsKeyDown in order to make a WINCMD program act on a hotkey. For example, to have your WINCMD program leap into action at the press of Ctrl-F9, you could loop on the statement

```
If (IsKeyDown("{F9}") AND IsKeyDown
     ("{control}")) do

   hotkey stuff...

end
```

Since IsKeyDown and IsKeyShifted work on keys, not characters, not every character is accepted by these functions. You should make it a general rule to pass the unshifted character to these functions. In the case of letters of the alphabet, this is not a requirement, but for keys that contain two unrelated characters, such as the equal sign and the plus sign, use the unshifted character to specify the key.

The IsKeyShifted function determines whether a key has been pressed an odd number of times since the system booted up. While you can use this function to determine if the A key has been pressed an odd number of times, the real value of the function is to check the CapsLock, NumLock, and Insert keys. This function will return a nonzero value if the number of keystrokes is odd.

The ShowWindow function allows a WINCMD program to show or hide a window on the desktop. The first parameter to this function is the handle of the window on which to operate. The second and final parameter is a flag: either WINDOW_SHOW or WINDOW_HIDE. To make a window invisible on the desktop, the command would be

```
ShowWindow (handle, WINDOW_HIDE)
```

ShowWindow is handy if you want your WINCMD program to be stealthy. With ShowWindow, you can hide your WINCMD window when it's waiting for an event or busy performing a task.

The SetWindowText function allows a WINCMD program to set the title text of a window. The main use of this function is to change the title text of the WINCMD window, but it can be used to set the title text of other windows on the desktop. The two parameters to this function are the window handle and the new title text. For example, to change the title bar of the WINCMD window to "Hi Bob!", the command would be

```
SetWindowText (hMain, "Hi Bob!")
```

Notice the use of the predefined variable hMain, which always contains the handle of the WINCMD window.

The IsRunning function indicates if a particular program is currently running on the desktop. The function has one argument—the name of the program—and returns a nonzero value if that program is running. The program name can either include a path or not. If a path is included, the program and path must be an exact match to the program running. If no path is specified, any program whose name matches the filename will be detected. The wildcard characters * and ? can be used in the program name but not in the path.

For example, the command IsRunning ("winword.exe") will return a nonzero value if Microsoft Word for Windows is running. Notice that the example did not specify a path. Had a path been specified, as in

```
IsRunning ("d:\editors\winword
          \winword.exe")
```

the function would only find WinWord if the .EXE file was in the D:\EDITORS \WINWORD directory. The example could have specified the extension .EXE by using the call IsRunning ("winword.*"). But in the unusual case that a program such as WINWORD.COM was running in a DOS box, the function would return "found" even if WINWORD.EXE was not running.

IsRunning is smart enough to check the names of programs running in DOS boxes. But if a DOS box is running a program with a .PIF file, the name of the .PIF file will be checked instead of the name of the program that the .PIF file is running. Similarly, if a batch file is used to launch the program, the batch filename is the one checked, not the program started by the batch.

IsRunning is powerful when used with the new LaunchSpecial statement in WINCMD 1.3. With IsRunning, a WINCMD program can determine what programs are and aren't running when it starts. This lets you check for a program, and then launch it if necessary.

The GetDate and GetTime functions return the current date and time. Neither of these functions take any parameters. The date and time are returned in the *mm/dd/yy* and the *hh:mm:ss* formats. Hours are in 24-hour format, so if it is 6:00 P.M., GetTime will return 18 in the hours field. Both functions display leading 0s for single digits. For example, March 2, 1994, would be displayed as 03/02/94. This allows easy parsing of the date and time values using the SUBSTR function.

The last two functions new to WCL2—ReadINIData and WriteINIData—provide WINCMD programs with the ability to read and write .INI file data. ReadINIData takes four parameters: the section name in the .INI file to be read, the name of the variable to be read (the *key*), the default value that will be returned if the section or key cannot be found, and the name of the .INI file itself. WriteINIData also takes four parameters: the section name, the key name, the value to write to the .INI file, and the name of the .INI file to write.

The primary purpose of .INI files is to save data when a program is not running, but these files do have other uses. Since Windows keeps much of its configuration data in WIN.INI and SYSTEM.INI, these WCL2 .INI functions can be used to read and modify your Windows configuration. Of course, you will want to be extremely careful when modifying your Windows configuration, since writing the wrong values to these critical files can be disastrous to your Windows setup!

USING WCL2 WITH WINCMD AND WCLIB Now that there are two function libraries—WCLIB and WCL—and four versions of WINCMD (Versions 1.0, 1.1, 1.2, and 1.3), WINCMD program writers need to take care to detect what environment their programs are running in. Since WINCMD 1.1, a predefined variable called WINCMDVER has contained the version number of WINCMD. To include a test for 1.0, which did not have this variable, a short function such as the following can be written:

```
GetWINCMDVer:
   if (WINCMDVER = "WINCMDVER")
      WINCMDVER = 10
   return WINCMDVER
```

the function would then be called this way:

```
version = GetWINCMDVer()
```

You can determine if a function library is present by using the LIBVER command. LIBVER's single parameter is the name of a function library, such as WCLIB or WCL2. The function returns the version number of the library or 0 if the library is not loaded.

An example of a routine that uses these variables and functions to verify the necessary components is shown in Figure 2.

```
//--------------------------------------------------------------------------
// CheckVers - Checks the necessary version WINCMD numbers.  Returns
// 0 if all versions ok.
// arg(1) - Name of program
// arg(2) - required version of WINCMD
// arg(n+3) - Name of required function library
// arg(n+4) - required version of library passed in arg (n+3)
//--------------------------------------------------------------------------
CheckVers:
   if (WINCMDVER < arg(2)) do
      errver = "version "+(arg(1) / 10)+("."+arg(1) mod 10)
      MsgBox ("You need "errver" of WINCMD.EXE to run "+arg(1), "WinCmd",0)
      return 1
   end
   count = 3
   while (count < arg()) do
      if (LibVer(arg(count)) < arg(count+1)) do
         errver = "version "+(arg(count+1) / 10)+("."+arg(count+1) mod 10)
         MsgBox ("You need "+errver+" of the WinCmd library "+arg(count)+" to run "+arg(1),"WinCmd",0)
         return 1
      end
      count = count + 2
   end
   return 0
```

Figure 2: **CheckVers is a routine that checks the version of WINCMD, as well as the versions of the WINCMD add-on libraries.**

The routine, called CheckVers, takes a variable number of arguments; the first is the name of the WINCMD program being executed. The program name can be determined in the main body of a WINCMD program with the call Arg(0). The second parameter passed is the required version of WINCMD. To require WINCMD 1.3, for example, pass the value 13 in this parameter. The remaining parameters are passed in pairs—first the name of the required library, and then the version of the necessary library. In this example, we require WCLIB 1.0 and WCL2 1.0, so the call to CheckVers would look like

```
if (CheckVers(arg(0), 13, wclib,
    10, wcl2, 10))
   exit 1
```

Since CheckVers returns 1 if any of the components cannot be found, a simple IF statement can be used to detect this condition, and you can end the program with an EXIT statement.

TRASHCAN DEMONSTRATION The example program TrashCan (distributed with WCL2) showcases many of the new functions of WCL2 and the enhancements to WINCMD. TrashCan is a WINCMD program that implements the classic drag-and-drop trash can popularized by the Macintosh. Files dropped on the TrashCan icon are moved to a Save directory. TrashCan never deletes the files in the Save directory, although the program could be modified to do so.

At about 250 lines, TrashCan is a moderately complex WINCMD program that centers around the new OnDrop statement. The new LaunchSpecial statement is used to synchronize running the DOS commands that perform the critical functions of TrashCan. Various other new WCL2 functions are used to aid in filename parsing and other necessary tasks.

TrashCan begins with a call to the same CheckVers function discussed earlier. If all the files are up to date, TrashCan calls the GetSaveDir function to get the path of the Save directory. GetSaveDir attempts to read the directory from the TRASHCAN.INI file. If the .INI file does not exist, a dialog box is presented to the user to ask for the directory. Once the directory is returned from either the user or the .INI file, a quick check is made to ensure that the directory exists.

The technique used to check for a directory is borrowed from *PC Magazine*'s own batch-file guru, Neil Rubenking. All that is required is to append the name of the null device (NUL) to the end of the path and check to see if DOS thinks the resulting filename exists. Instead of the IF EXIST line used in standard .BAT files, GetSaveDir uses the FILEEXIST function available in WCLIB. If the directory read from the .INI file does not exist, or if the directory returned from the user could not be created, GetSaveDir displays an error box and terminates the program. This technique works under plain, nonnetworked DOS. Under DR DOS 6.0 and some network operating systems, the FILEEXIST test will always return TRUE for the null device even if the directory does not exist.

The heart of TrashCan is the short loop

```
// Main program loop
while (1) do
   // Fill in output box with list
   // of current files in trash dir
   count = GetSaveFileCnt(TrashDir)
   // Set drop icon
   if (count)
      seticon (hMain, hIconF)
   else
      seticon (hMain, hIconE)
   // Wait till something's dropped
   stop
end
```

In this loop, TrashCan calls the GetSaveFileCnt routine to determine if any files are in the Save directory. Depending on whether any files are in the directory, one of two icons are displayed. After the icon or icons have been set, the program halts with a STOP statement.

The STOP statement is a bit deceiving. Unlike an EXIT statement, STOP does not terminate the program; instead, the program is stopped in place. Stopping a WINCMD program is much better than a continuous loop, because a stopped WINCMD program does not take up any valuable processor time.

Even with the program stopped, the OnDrop statement will still take control and execute the remainder of its line if a file is dropped on the TrashCan icon. When a file is dropped, WINCMD executes the remainder of the OnDrop line—in this case, the call to CopyFunc. When CopyFunc returns, WINCMD continues execution at the statement after the STOP. The WHILE loop returns WINCMD to check the number of files in the Save directory, and then executes the STOP statement to await another file drop.

CopyFunc's job is to query the names of the dropped files and directories using the GetDropFile function, and then move them into the Save directory. If the returned name is a file, the process is simple: Just copy the file into the Save directory and erase the original. This function is performed by the function MoveFile. Users of DOS 6.*x* can modify MoveFile to use a single MOVE statement instead of the COPY and DEL used in this example.

If the dropped item is a directory instead of a file, the process is a little more complex. For a directory, TrashCan must determine the names of each file and subdirectory inside the directory to be moved, and then process each file and directory individually. This is where the LaunchSpecial statement comes into play. It is standard batch file practice to redirect the output of a DOS command to a file, and then have the batch file process the redirected data contained in that file. Before WINCMD 1.3 and the new LaunchSpecial statement, this was not easy to do, since WINCMD did not wait for a DOS command (or a program) to complete before continuing on to the next statement in the WINCMD program. With LaunchSpecial, a WINCMD program will wait until a task is complete. In this case, the MoveDir function redirects the DIR command output to a file, and then reads that file (using repeated calls to the WCLIB function FileReadLine) to determine each of the filenames in the directory.

A problem arises when the directory being moved contains subdirectories. When a subdirectory is found, we need to call the MoveDir function from *within* MoveDir. The technique in which a function calls itself is called *recursion*, and while incredibly powerful, it is difficult to implement—especially in a language such as WINCMD where all variables are global.

Looking carefully at the MoveDir function of TrashCan, you can see that while I use a number of variables as temporary place holders, the only variable that is needed both before and after the recursive call to MoveDir is the handle of the redirected data file—appropriately named Handle. The problem is that if MoveDir calls itself, the value in Handle is overwritten by the second call. To avoid this, the following little trick is used to save the value of Handle in a string called rArray:

```
//Save handle as string in rArray
rArray = "#"+handle+"$"+rArray
//Recurse calling this routine again
rc = movedir (filename)
//Restore variables
rArray = substr (rArray, 1, 1000)
//Strip off leading #...
//Get back handle value
handle = substr (rArray, 0,
instr(rArray,"$"))
//Strip handle from rArray
rArray = substr (rArray, instr
(rArray,"$")+1,1000)
```

The key to this trick is that WINCMD does not differentiate between numbers and strings. A number can be appended to a string, and conversely, the numeric value of a string can be used in any mathematical operation. In the code fragment above, the plus signs are actually appending parts of a string that is then assigned to rArray. The SubStr function is used to break apart a string into smaller substrings.

The rArray variable is nothing more than a string that contains the recursive values of Handle, each value separated by the characters # and $. (I could have used any nondigit characters to separate the Handle values.) Once Handle is saved in rArray, MoveDir can safely be called. If the recursive call to MoveDir must move a directory, and therefore call itself a second time, the new value of Handle is appended to rArray. The old value is not lost, because Handle is not assigned to rArray; the old value is *appended* to the existing value in rArray.

When the recursive call to MoveDir returns, Handle is restored by taking the first number off the beginning of the rArray string. Since each number is separated by a space, the InStr function can be used to return exactly the number of digits in Handle. Once Handle is restored, that number is stripped from rArray using the SUBSTR and InStr functions. The use of rArray allows any number of recursive calls to MoveDir.

TrashCan demonstrates only one of the many new programs that can be built using WCL2 and WINCMD 1.3. Enjoy WCL2 and the new functions of WINCMD 1.3. I already find them indispensable in my WINCMD programs. □

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

GUIDE TO OUR UTILITIES

WCL2

PURPOSE: WCL2—WINCMD Library 2—is a second add-on library for WINCMD, *PC Magazine*'s batch-language utility for Windows. WCL2 can be used in conjunction with WCLIB, the first add-on library. WCL2 requires Version 1.3 of WINCMD.

SETUP: To upgrade to the new version of WINCMD, simply copy the new WINCMD.EXE over the old one. To install WCL2, copy the file WCL2.WCL into the same directory.

REMARKS: WCL2 contains 21 new WINCMD functions in six different groups. Four new String functions—Val, Chr, InStr, and Reverse—let you convert a character to its ASCII code and back again, locate substrings within a string, and reverse the order of a string. The six filename functions—FullPath, GetFileDrive, GetFileDir, GetFilePath, GetFileName, and GetFileExt—return the full pathname of a file or any component of the filename. You can obtain the device, path, filename, or file extension of a file.

The two file-processing functions—FileOpenBox and FileSaveBox—let you access the standard Windows dialog box for opening and saving files.

The two keyboard-monitoring functions let you detect whether a key is being pressed and whether it is in a shifted state.

The three Window functions—SetWindowText, ShowWindow, and IsRunning—let you set the title text of a window and let you make a window visible or invisible. The IsRunning function lets you detect whether a specified program is loaded in memory.

The GetDate and GetTime functions return the system date and time in the formats *mm/dd/yy* and *hh:mm:ss.* The GetTime function uses a 24-hour clock.

The .INI file functions—ReadINIData and WriteINIData—let you read and write to .INI files from your WINCMD programs, locating the appropriate sections and keys.

& HOW TO GET THEM

PC Magazine's utilities are available by modem from PC MagNet, our on-line service. PC MagNet is part of ZiffNet, a group of on-line services from Ziff-Davis Publishing Company. ZiffNet is hosted by CompuServe Information Service.

THE FIRST TIME If you've never used ZiffNet, you'll need to find the access number nearest you. Set your communications software to 7 data bits, even parity, no stop bits, 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 closest access number. Or dial 800-635-6225 (voice) for instructions.

HOW TO JOIN ZIFFNET Dial the local access number. At the HOST NAME prompt, enter CIS. At the USER ID prompt, type 177000,5000. When you see the PASSWORD prompt, enter PC*MAG-NET. Finally, when you see the ENTER AGREEMENT NUMBER prompt, enter PCMAG94. Current CompuServe members can type GO PCMAG at any CompuServe ! prompt. Once you enter your billing information, you'll see your personal user ID and temporary password.

MEMBERSHIP PRIVILEGES Membership costs $2.95 per month. You receive free access to *PC Magazine* Editors' Choice awards, Product Reviews Index, weekly news from *PC Week*, Buyer's Market, ZiffNet Highlights, the Support Forum, and the current utility (type GO ZNT:FREEUTIL). For other services, connect-time charges vary according to modem speed. Billing is based on 1-minute increments. You can also download all of *PC Magazine*'s utilities and tips. Type GO ZNT:TIPS.

Prodigy members can download the utilities from ZiffNet selections on Prodigy for a transfer fee of $3 per program. Jump 'ziff select' and browse the computing topics for programs.

FREE UTILITY You can download this month's utility, free of charge, from *PC Magazine*'s BBS by dialing 617-231-4450 with your modem. Just enter your name at the prompt and follow the menus.

NO MODEM, NO PROBLEM If you don't have a modem, you can get our utilities by mail. Just send your request with your name, address, and preferred disk size to Public Brand Software, P.O. Box 51315, Indianapolis, IN 46251; fax, 317-856-2086.

COPYRIGHT INFORMATION The programs presented in *PC Magazine* can be copied but are copyrighted. You may make copies for others if no charge is involved, but making copies or using any portion for any commercial purpose is strictly prohibited.