# Windows Programmer's Journal Vol. 1 No. 5 May 1993

```
---------------------------------------------------------------
The Windows Programmer's Journal                Volume 01
Copyright 1993 by Peter J. Davis                Number 05
and Mike Wallace                                May 93
---------------------------------------------------------------
```

A monthly forum for novice-advanced programmers to share ideas and concepts about programming in the Windows (tm) environment. Each issue is uploaded to the info systems listed below on the first of the month, but made available at the convenience of the sysops, so allow for a couple of days.

You can get in touch with the editors via Internet or Bitnet at:

HJ647C at GWUVM.BITNET or HJ647C at GWUVM.GWU.EDU (Pete)
CompuServe: 71141,2071 (Mike) 71644,3570 (Pete)
AmericaOnline: PeteDavis
Delphi: PeteDavis
GEnie: P.DAVIS5

or you can send paper mail to:

Windows Programmer's Journal
9436 Mirror Pond Dr.
Fairfax, Va. 22032

We can also be reached by phone at: (703) 503-3165.

The WPJ BBS can be reached at: (703) 503-3021.
The WPJ BBS is currently 2400 Baud (8N1). We'll be going to 14,400 in the near future, we hope.

Table of Contents

Windows Programmer's Journal Staff:

Publishers ......................... Pete Davis and Mike Wallace
Editor-in-Chief ................... Pete Davis
Managing Editor ................... Mike Wallace
Contributing Editor ................ Dave Campbell

Contributing Writer ................ Dave Campbell
Contributing Writer ................ Eric Grass
Contributing Writer ................ Loewy Ron
Contributing Writer ................ Rodney Brown

## *Legal Stuff*

- Microsoft, MS-DOS, Microsoft Windows, Windows NT, Windows for Workgroups, Windows for Pen Computing, Win32, and Win32S are registered trademarks of Microsoft Corporation.

- Turbo Pascal for Windows, Turbo C++ for Windows, and Borland C++ for Windows are registered trademarks of Borland International.

- WordPerfect is a registered trademark of WordPerfect Corporation.

- Other trademarks mentioned herein are the property of their respective owners.

- WPJ is available from the WINSDK, WINADV and MSWIN32 forums on CompuServe, and the IBMPC, WINDOWS and BORLAND forums on Genie.   It is also available on America Online in the Programming library.   On Internet, it's available on WSMR-SIMTEL20.ARMY.MIL and FTP.CICA.INDIANA.EDU.   We upload it by the 1st of each month and it is usually available by the 3rd or 4th, depending on when the sysops receive it.

- The Windows Programmer's Journal takes no responsibility for the content of the text within this document. All text is the property and responsibility of the individual authors. The Windows Programmer's Journal is solely a vehicle for allowing articles to be collected and distributed in a common and easy to share form.

- No part of the Windows Programmer's Journal may be re-published or duplicated in part or whole, except in the complete and unmodified form of the Windows Programmer's Journal, without the express written permission of each individual author. The Windows Programmer's Journal may not be sold for profit without the express written permission of the Publishers, Peter Davis and Michael Wallace, and only then after they have obtained permission from the individual authors.

# WPJ.INI

By Pete Davis

Another month, another issue. Wow, up to number 5. I guess I say something along those lines every month, but the response just keeps getting better and better and I guess I'm just surprised that we're actually getting these things out every month.

I received some questions from our readers about some of the articles in the magazine. Their concern was that we seem to try to fit a lot of information into short articles and that sometimes it's not in-depth enough. Well, to those who think that, let me explain. When we write articles for the magazine (and I can only speak for me and Mike), we do try to fit the information into rather short articles and we don't go into a great deal of depth a lot of the time. The reason for this is that Mike and I both work full-time. On top of that, we do several other things outside of work and the magazine. Writing in-depth articles means lots of research and time. We can't afford that kind of time, unfortunately. It is my hope that, someday, I'll have more time to spend writing more in-depth articles. Until that day, we can only offer what we can. We try to give as much information as is applicable and when a readers asks us to clarify something, we try our best to do that. If you ever feel like an article needs clarification, let us know and we'll either do that directly, via mail, or we'll do an addendum in a future issue.

Next subject: Me! After the last paragraph, now's the time to spill the beans about why things are going to be changing a bit for me. I am going to be writing a book for Addison-Wesley publishing on Windows programming (NT and the 32-bit stuff). Unlike the writing I do in the magazine, the stuff in the book will be the kind of writing that requires months of research and lots of checking and double-checking (like I said, we don't have the time to do that for the magazine). Anyway, the book is on Win32 programming and it will be in the Andrew Schulman Programming Series. I have to say I'm awfully excited about the whole thing and I'm really looking forward to it. I couldn't possibly thank Andrew Schulman enough for the chance he's giving me. So, that's what's up with me. What does this mean? Well, for one thing, all of my articles are going to be on Win32 programming for the next few months. I've got to do this just to help keep my mind trained on the topic, if nothing else. Also, I might not be able to do quite as much writing for the magazine as I'd like.

Also, nothing firm here, but it looks like I might be doing an article with Ron Burk of Windows/DOS Developer's Journal. This, like the book, is a sort of first for me and I'm really excited about that. I'll give more info on this next month, when I know more.

So, that's what's going on with me. After the book is done, perhaps I'll start writing about things other than Win32, but hopefully, by then, EVERYONE will be programming in Win32. It could happen! By the way, I expect each and every one of you to buy my book when it's done.

We've got an article by a guy named Eric Grass in this issue. I've got to talk about how I met Eric for a minute. Eric writes a shareware Windows disassembler called WDASM. It's a really nice product and I felt I had to review it in this issue. Anyway, when I registered WDASM (registration is a mere $10, I felt obligated to pay $20, because it's worth at least that) I sent in some suggestions for expanding WDASM. In our ensuing e-mail volleys, I asked Eric if he'd write an article for the magazine. Anyway, read his article on writing an editor in Windows and read my review of his disassembler and then get his disassembler! You'll thank me for recommending it and you'll thank Eric for writing it.

The last thing I want to talk about is submissions. The number of article submissions seems to be on the decline. Unless we start getting more articles, this magazine might have to become a bi-monthly affair. I know that I won't have as much time as I'd like to do all the work required to keep it monthly. Mike, I'm sure, will do his best to write for it, but unless we get more submissions, I can't see us staying at a monthly pace. The final decision will have to be Mike's since he is going to be stuck with doing most of the work for the next few months.

Remember, we don't do this just for us, we do it for you, the readers. If you've felt like you've gotten something out of the magazine, try to return the favor and give a little of your time to help keep it a monthly magazine. I know a lot of you want to see it stay monthly. I know I do. Writing an article can take as little as a few hours. Most of you have something you've learned somewhere that could help a lot of other people out. We're counting on you to share that information.

Until next month, peace.

P.S. I'm writing this the day of the release of this issue to apologize to our readers. Things have been incredibly hectic lately for me because of the book and now this possible article with Ron Burk. The timing on the magazine was just off a bit. Mike has also been really busy, so neither of us had the time this month that we'd normally like to put into the magazine. Please accept our apologies.

## ***Beginner's Column***

By Dave Campbell

Don't you just hate it when you look at a ton of Windows Applications, and they all look alike, and they all do the same things in response to the same keystrokes, and YOU DON'T HAVE ANY IDEA HOW THEY DID IT? That's how I was about file open dialog boxes. I knew I could do it, I just didn't want to. Yeah, right. So, this month, I am going to give one possible example of a way to do one (the hard way), and along the way, we'll learn some things about list boxes.

First I want to pass along a few things. The code out of which I stripped the file open dialog box uses the Borland classes for the dialog box displays. I have been trying my best to stay away from any compiler-particular code for my articles. Consequently, I was mildly taken aback when my dialog box came up with the Borland background in place. How could this be? Well...I run a background menu processor that is in Beta at this time, (waiting on documentation), and it uses the Borland classes. Since that is all instantiated in memory, when our new Hello program came in, and tried using the Borland calls, it all worked. Basically I got lucky. Usually it works the other way. You forget to put it in, and then try to call it.

My reason for mentioning all this is: if you are in a debug cycle, and going into and out of Windows relatively often, remember that if you hold a shift key down during Windows start-up, you won't get your "run=", "load=", or "Startup" files. All you'll get is Windows, and you can then make sure that if you need a DLL, you load it yourself.

The second thing is to be sure to read my review of the Microsoft Developer's Network program in this issue. This is like the mother-load of Windows help.

Now on to a file open box. What I want to do is display a dialog box that has a file spec in it, for example "*.EXE", and various areas to display paths, and file names, and directory names, and a way to select a file or quit. That will just about solve all the requirements (not counting drag-and -drop). How this is implemented in my other application is: in a dialog box is a text box requesting a file name. If OK is selected, and the file name is NULL, then I pop up the file open box. In Hello, we already have a File Open menu choice, and are going to use it.

First let me display what this dlg code looks like:

From Hello.DLG
------------------------

```
FILEOPEN DIALOG 80, 60, 148, 101
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION
CAPTION "Find File"
FONT 8, "Helv"
BEGIN
     CONTROL "File name:", -1, "STATIC"
      SS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP, 2, 4, 144, 10
```

```
        EDITTEXT IDD_FNAME, 2, 14, 144, 12, ES_AUTOHSCROLL
        CONTROL "Files in", -1, "STATIC"
          SS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP, 2, 30, 38, 12
        CONTROL "", IDD_FPATH, "STATIC"
          SS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP, 48, 30, 98, 12
        LISTBOX IDD_FLIST, 2, 43, 82, 58, WS_TABSTOP | WS_VSCROLL
        CONTROL "Select", IDOK, "BUTTON",
          BS_DEFPUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP,
                                                    90, 52, 56, 14
        CONTROL "Cancel", IDCANCEL, "BUTTON", WS_GROUP, 90, 76, 56, 14
```

The first new item to us is the EDITTEXT line. EDITTEXT creates a child window of the EDIT class which gives us a place to allow a user to type some text for us to capture. The user may move from edit box to edit box, and get the focus by clicking the mouse inside it, or we as developers can place the user there. In our example, we will place the user in the box. When the user sees a flashing prompt in the text box, it is safe to type and edit the typing. As with any Windows text box, the mouse may be used, along with the backspace key to provide editing features. All this for one line in the dialog box!! Isn't it simple to program in Windows?

As with most Windows features, the parameters are many. The only one used in this example is ES_AUTOHSCROLL. This does exactly what it sounds like. If the text scrolls beyond the right-hand margin during input, the box will scroll with it. The default options are: ES_LEFT | WS_BORDER | WS_TABSTOP.

## LISTBOX
--------------

The next new control is LISTBOX. A listbox is a child window containing a list of character strings, allowing the user to scroll and make selections. When a string is selected with a mouse click, the string is highlighted, and a message sent to the parent window. I have chosen to allow a vertical scrollbar on our listbox, and adding it to the style list is all it takes to do that.

Invoking the dialog box is the easy part. In place of the message handling of last month in response to the File Open menu, we'll do:

```
if (DoFileOpenDlg(hInst, hWnd, &of))
     lstrcpy(OutMsg, (char far *)pof->szPathName);
```

Notice the file open dialog box is embedded in an if statement. In this circumstance, we are going to check the return value of the dialog box ourselves, and if we return TRUE, we know we have a good value for the file, and we display it in the same manner the other data is displayed from last month.

Don't be confused as to the parameters passed in this statement. 'DoFileOpenDlg ' is NOT a dialog box call. This is simply a launcher for the dialog box that we are setting up with some parameters, and from which we are going to get a return value.

The real dialog box code inside DoFileOpenDlg looks like our dialog boxes from the last two issues:

```
lpfnFileOpenDlgProc = MakeProcInstance((FARPROC)FileOpenDlgProc, hInst);
iReturn = DialogBox(hInst, "FileOpen", hWnd, lpfnFileOpenDlgProc);
FreeProcInstance(lpfnFileOpenDlgProc);
```

The only difference here is picking up the return value from 'DialogBox'.

OFSTRUCT
-------------------

At this point, however, I need to discuss one of the parameters to DoFileOpenDlg, and that is &of. of is listed way up in the beginning of Hello.C as being of type OFSTRUCT. OFSTRUCT is defined as:

```
typedef struct tagOFSTRUCT {      /* of */
    BYTE   cBytes;
    BYTE   fFixedDisk;
    UINT   nErrCode;
    BYTE   reserved[4];
    BYTE   szPathName[128];
} OFSTRUCT;
```

This structure is the result of opening a file. As you can see, this structure tells us if the file is on a hard disk, if there was an error opening the file, and the complete pathspec for the file.

DoFileOpenDlg
------------------------

Inside DoFileOpenDlg, a few global variables are set:

```
pof = pofIn;
lstrcpy(szFileSpec, "*.EXE");
lstrcpy(szDefExt,    "EXE");
wFileAttr = DDL_DRIVES | DDL_DIRECTORY;
```

1) 'pof' is defined above with of as an LPOFSTRUCT
(or Long Pointer to an OFSTRUCT).
2) *.EXE is picked as the default filespec (to begin with).
3) the default file extension of EXE is set.
4) the global file attribute variable, wFileAttr is set to
DDL_DRIVES | DDL_DIRECTORY which declares that we are interested in files that are drive or directory names (to begin with).

Windows message Loop
--------------------------------------------------

With all that out of the way, let's start with the WM_INITDIALOG case:

```
    case WM_INITDIALOG :
        SendDlgItemMessage(hDlg, IDD_FNAME, EM_LIMITTEXT, 80, 0L);
        DlgDirList(hDlg, szFileSpec, IDD_FLIST, IDD_FPATH, wFileAttr);
        SetDlgItemText(hDlg, IDD_FNAME, szFileSpec);
        return TRUE;
```

Remember, this is the first thing that happens to the dialog box. Right away we jump into three commands that haven't been discussed before. Two are related, and I'll discuss them first:

```
   SendDlgItemMessage(hDlg, IDD_FNAME, EM_LIMITTEXT, 80, 0L);
```

SendDlgItemMessage sends a message to a specific control in a dialog box, identified by, in this case, the IDD_FNAME control ID. SendDlgItemMessage does not return until the message has been processed. 'EM_LIMITTEXT, 80' sets the control up for receiving up to 80 characters of text, and the 0L is an empty lParam value for the call. The bottom line is that this call did one thing: it set the length of the text box to be 80 characters.

```
   SetDlgItemText(hDlg, IDD_FNAME, szFileSpec);
```

SetDlgItemText sets the title or text of a control in a dialog box. In our case, it is the same control as the last call, and this time we send it the "*.EXE" default file spec from above.

Now the fun one:

```
    DlgDirList(hDlg, szFileSpec, IDD_FLIST, IDD_FPATH, wFileAttr);
```

DlgDirList fills a list box with a file or directory listing. What gets displayed is controlled by the parameters passed in: szFileSpec and wFileAttr, both discussed above. IDD_FLIST is the ID of the listbox control itself, and IDD_FPATH is the ID of a static control that will be used by Windows to display the current drive and directory. If this is 0, no display will be done.

If the filespec is a directory name (or a null string), the string will be changed to "*.*". After the box is filled, the filespec is stripped of any drive or path information.

At this point, the dialog box is up, and the listbox is full of file listings, and the user can move around in there just like any other Windows file open box.

## Mouse Messages
---------------------------

This brings us to the fun part of the dialog box code. Of course we want OUR dialog box to be just like everyone elses, so double clicks select, etc. But then we have to write that code! So here goes(taken liberally from Charles Petzold's Great book "Programming Windows"):

```
    case WM_COMMAND :
        switch (wParam)
            {
```

```
            case IDD_FLIST :
                switch (HIWORD(Lparam))
                    {
                    case LBN_SELCHANGE :
                        ----code below----
                        return TRUE;

                    case LBN_DBLCLK :
                        ----code below----
                        return TRUE;
                    }
                break;
```

WM_COMMAND is the workhorse message of Windows. A window receives a WM_COMMAND just about anytime the user does something. If the user selects a menu item, or a child control sends a note back to 'mom'. In our case, we are concerned about messages coming from the list box, so we look for IDD_FLIST, our ID for the listbox.

If we receive a message from IDD_FLIST, we are likely to get two things: 1) we select an item by single-clicking it with the mouse, or 2) we double-click something to not only select it, but SELECT it (in other words, do something with it now, I only have enough time to deal with the mouse).

This kind of information is passed to the window in the high-order word of Lparam. This is officially called 'wNotifyCode'. If the message is from a control, this will be a Windows-type message. If the message is from an accelerator, wNotifyCode is 1, and if from a menu, it is 0.

This is why there is a secondary switch case inside the outer one. Now we have to deal with what is going on with the listbox. Consequently, we have our two messages: 1) LBN_SELCHANGE, and 2) LBN_DBLCLK.

LBN_SELCHANGE
-----------------------------

```
    if (DlgDirSelect(hDlg, szFileName, IDD_FLIST))
        lstrcat(szFileName, szFileSpec);
    SetDlgItemText(Hdlg, IDD_FNAME, szFileName);
```

DlgDirSelect assumes that the listbox identified with the ID (in our case IDD_FLIST), has been filled with a DlgDirList function (good thing we did that), and returns us the selection currently selected. The second parameter should be a pointer to a 128-byte character string (to be fer sure, fer sure). By the way, I will change this to 128 bytes before anybody else sees it, now that I have looked that one up.

The function returns non-zero if successful, zero otherwise. And, the string stuffed into, in our case szFileName, will be a drive letter, or a directory name, with any brackets, etc. removed. The function appends a ':' to a drive letter and a '\' to a subdirectory name.

So, when we start out, szFileSpec is *.EXE,   and if we select [-c-] for example, in this loop we concat c: with *.EXE, and send C:*.EXE to IDD_FNAME with the SetDltItemText call that is next. Notice this doesn't do anything more than save the names in the global strings, and display them in the text box. However:

LBN_DBLCLK
---------------------

```
      if (DlgDirSelect(hDlg, szFileName, IDD_FLIST))
          {
          lstrcat(szFileName, szFileSpec);
          DlgDirList(hDlg, szFileName, IDD_FLIST, IDD_FPATH, wFileAttr);
          SetDlgItemText(hDlg, IDD_FNAME, szFileSpec);
          }
      else
          {
          SetDlgItemText(hDlg, IDD_FNAME, szFileName);
          SendMessage(hDlg, WM_COMMAND, IDOK, 0L);
          }
```

The first if message is identical. If we get a good return from DlgDirSelect, concat the name, and then...oh yeah, now we need to do something, since we double clicked!! Ok, so what would we like it to do? Most of us would agree that leaving the szFileSpec alone,   *.EXE, and changing to the drive or directory, in addition to changing the text in IDD_FNAME would be nice.

So that's what we do. The second half is identical to what we did in the LBN_SELCHANGE message, just display, but the first part is handled by re-executing the DlgDirList call, only passing it the new szFileName we concatenated.

This is all well and good if we double click on a directory or a drive letter, but what if we double click a file name? That's what the else case is for. Remember above I said that a good return from DlgDirSelect gave us a drive or subdirectory name? Well, a FALSE return gives us a filename. So, if we take the 'else' clause, for FALSE, we know that szFileName is the filename that the user double-clicked, and we stuff that into the IDD_FNAME box, followed by sending OURSELVES (!!) an IDOK message.

IDD_FNAME
-------------------

IDD_FNAME is also user-modifiable, so if the user types something in there, we want to know what is going on, so we handle calls originated from there:

```
    if (HIWORD(lParam) == EN_CHANGE)
         EnableWindow(GetDlgItem(hDlg, IDOK),
            (BOOL) SendMessage(LOWORD(lParam), WM_GETTEXTLENGTH, 0, 0L));
```

Remember above in talking about messages from child controls, how the message is passed in the high-order word of lParam?, well here we go again. And, this time we are looking for a changed text box, or EN_CHANGE. If this is the case, Windows has already updated the text in the box.

The EnableWindow line is a combination of three windows functions, and as such is extremely messy. The EnableWindow call really looks like:

```
    EnableWindow(hwnd, fEnable),
```

where hwnd is the handle of the window to be enabled or disabled according to the Boolean equivalence of fEnable (ie, TRUE/FALSE).

The window we want to enable/disable is the one whose handle is returned from GetDlgItem(hDlg, IDOK). hDlg is the handle to our dialog box, so that's easy, but how about the IDOK? Go back to the .DLG file and look at what happens if the user clicks the button "Select". The button returns IDOK. It works the same as an OK button, but has Select written on it instead. That is the window whose handle we are going after for the EnableWindow call.

Now, what are we going to do with it?

```
    SendMessage(LOWORD(lParam), WM_GETTEXTLENGTH, 0, 0L)
```

SendMessage is going to send the command WM_GETTEXTLENGTH to the window whose handle is the LOWORD of lParam. The high word was discussed above (twice) as the message sent, the low word is the handle of the control sending the message. Therefore, SendMessage is going to send a WM_GETTEXTLENGTH command to the IDD_FNAME box which will return the length, in bytes of the text in IDD_FNAME.

Whew...let's see now: if we square the result, oops wrong article.

But, it's almost that complicated. The bottom line is we are going to send an EnableWindow message to the Select button only if there is text in the IDD_FNAME box after it has been modified. This will end up working its way back to us as an IDOK, and will select the filename.

Now wasn't that fun?

IDOK
--------

The IDOK message handler does the following:

1) Gets the filename from IDD_FNAME. We put it there, remember?
2) If szFileName ends in ':' or '\', append szFileSpec to szFileName.
3) If szFileName ends in a wildcard, execute DlgDirList with the new szFileName, and clean up the displays appropriately:
    a) copy szFileName to szFileSpec (since it had a wildcard anyway)
    b) write the new szFileSpec into IDD_FNAME
    c) if DlgDirList fails, beep to the user
    d) exit the message loop

4) If szFileName does not end in a wild-card, concat '\szFileSpec' onto

szFileName, and re-execute DlgDirList with this new filename.

5) If DlgDirList finishes successfully, displays are updated, and we exit the message loop

6) If DlgDirList does not finish successfully, it means that the szFileName did not contain a drive\directory structure. So, it is probably a file name. This is our back-door, so to speak, to get the file name.

7) If we've come this far, we assume we have a full file-spec on our hands.

8) We take two tries at opening the file with OF_EXIST set. This will open and close the file just to check for existence of the file. The OF_READ for read-only was thrown in as insurance. The first try is with the name we have already. If that fails, we try with the default file extension on it, just in case. If both fail, we beep to the user and exit.

9) If we are successful in opening and closing the file, we have finished what we set out to do, and exit the dialog box. The filename is copied from the OFSTRUCT way back up where the adventure started with File/Open, and at the bottom of that loop, we print execute a MessageBox call with the filename as the child window text.

I did gloss over one of Petzold's routines, and neglected to mention a couple of other things. Petzold used an internal routine called 'lstrchr' to find the existence of a character in a string. That's how we decided if there was a wild-card in the filename. He also used lstrcat and lstrcpy to let Windows handle the far pointer manipulation without messing up.

I know I've gone over the last part pretty quickly, but I tried to cover it completely. It's mostly 'C' code, and doesn't have a lot of Windows-particular stuff in it. I think I'd rather spend my time explaining the hairy Windows parts than mess so much with the 'C' parts.

## EndDialog
-----------------

That's it for now. As I have said, Please hang in there. If you are beyond the scope of this article, stick with me we are going to go places together. If you are way beyond this, write us an article. If you are bogged down, just compile it, and stare at the source. If you want help with something, send me a note.
Next month, unless I get mail not in favor of it, we're going to put this month's code into a DLL so we can use it later. Feel free to contact me in any of the ways below. Thanks to those of you that have e-mailed me questions, keep them coming. Notice that I now have an internet address, to make it real easy to get in touch with me. I want to rat out the things other people are having questions about, not just what I think people want to hear.

Dave Campbell
WynApse PO Box 86247 Phoenix, AZ 85080-6247 (602)863-0411
wynapse@indirect.com
CIS: 72251, 445
Phoenix ACM BBS (602) 970-0474 - WynApse SoftWare forum

ClickBar consists of a Dynamic Dialog pair that allows C developers for Windows 3.0/3.1 to insert a "toolbar" into their application. Microsoft, Borland, etc. developers may display a toolbar or tool palette inside their application, according to a DLG script in their .RC or .DLG file.

Borland developers may install ClickBar as a custom control, providing three custom widgets added to the Resource Workshop palette. These are three different button profiles defining 69 custom button images total. The buttons may be placed and assigned attributes identically to the intrinsic Resource Workshop widgets.

Source is provided for a complete example of using the tools, including the the use of custom (owner-drawn) bitmaps for buttons.

Version 1.5 uses single-image bitmaps to reduce the DLL size, and includes source for a subclass example for inserting a toolbar.

Registration is $35 and includes a registration number and printed manual.

| | |
|---|---|
| WynApse | |
| PO Box 86247 | (602) 863-0411 |
| Phoenix, AZ 85050-6247 | CIS: 72251,445 |

## ***Programming Techniques For Creating Text File Editors***

by Eric Grass

This article discusses advanced programming techniques for creating text editors under Windows. The Edit window class will not be implemented in order to focus on how to create more advanced types of editors. We will thus be creating a text editor "from scratch", so to speak. As an example, the source code for a simple text editor named EditPro has been included with this issue of WPJ. EditPro is a smaller version of another editor named KeyPro which I wrote recently.

EditPro simply loads and displays files using the Windows system font and does paging. This article covers these aspects of file editing, and then offers some tips on using the system caret.

The fundamental portions of the program (i.e., the WinMain function, etc.) were created using QuickCase:W for Windows. The source code contains some in-line assembly language code in order to improve the program's execution time in certain places where it is appropriate.

I compiled EditPro using QuickC for Windows. You may or may not get the following warnings when you compile:

```
qcw /AS /G2w /Zp /W3 /D_WINDOWS /Gi /Od /Zi -f A:\EDITPRO.C
A:\EDITPRO.C(190) : warning C4059: segment lost in conversion
A:\EDITPRO.C(191) : warning C4059: segment lost in conversion
A:\EDITPRO.C(195) : warning C4059: segment lost in conversion
A:\EDITPRO.C(198) : warning C4059: segment lost in conversion
A:\EDITPRO.C(200) : warning C4059: segment lost in conversion
A:\EDITPRO.C(208) : warning C4059: segment lost in conversion
A:\EDITPRO.C(254) : warning C4059: segment lost in conversion
linkw /ST:5120 /A:16 /CO @$$QCW$$.CRF
rcw -t EDITPRO.RES EDITPRO.EXE
cvpackw EDITPRO.EXE
EDITPRO.EXE - 0 error(s), 7 warning(s)
```

These warnings can be ignored. This occurs because the code extracts the offset from global memory pointers. I tried to figure out a way to use casting to avoid these warnings, but I couldn't.

The source code includes some helper routines (or, file routines) in a file named filerout.c. The procedure FileRoutine() merely invokes the Open dialog box and retrieves a file path name. The second two routines are used to merely display message boxes.

### Storing Text

The most important step in designing a file editor is designing the data structure for storing the lines of text of the file in memory. EditPro uses the following Line structure to store a single line of text:

```
    |¯                    ..........22222222222222
```

```
OFFSET:------|                111111111122..........44444444555555
             |_    0123456789012345678901..........23456789012345
             \/\/|\___________________________________________/
              \ \ \        \
               \ \ \        \__Text
                \ \ \________(BYTE) total number of bytes in line structure
                 \ \_________(WORD) memory handle of next line
                  \__________(WORD) memory handle of previous line
```

For each line we allocate between 5 and 256 bytes of global memory. If a line contains just N characters, then we only have to allocate N+5 bytes. Allocating more memory than this would be a waste of memory. Thus, this type of structure allows at most 251 characters per line. The first two bytes, bytes 0 and 1, store the memory handle to the previous line proceeding the current line. The next two bytes, bytes 2 and 3, store the memory handle to the next line following the current line. Thus, we may construct a linked list of lines. The byte at offset 4 stores the total number of bytes constituting the memory block minus 1, and must therefore be a value between 4 and 255. Finally, bytes 5 through 255 are used to store the actual text constituting the line. The primary motivation for selecting the value 256 as the maximum size of the Line Structure is so that we can store the size of the structure (at byte 4) using just one byte.

The file loading procedure can be expressed in pseudocode as follows:

```
.
.
.
Open the specified file;
if( size of file < MAXFILEBUFF)
    Allocate temporary file buffer of size = size of file;
else
    Allocate a temporary file buffer of size = MAXFILEBUFF;
Allocate and Lock a new Line Structure of size = 256 bytes;
Set Line Structure pointer to 0;
while( End of File not reached)
{   Read portion of file into temporary file buffer;
    Set file buffer pointer to 0;
    while( Not at the end of the temporary file buffer)
    {   if( file buffer pointer points to Newline character)
        {   Allocate and Lock new Line Structure of size 256 bytes;


            Store the handle of the previous Line in the new Line
            Structure;

            Reallocate the size of the previous Line Structure to
            N+5 bytes;

            Store the value N+4 at offset 4 in previous Line
            Structure;
```

```
                Unlock the previous Line Structure;
            }
            else
            {   Copy the character pointed to by file buffer pointer to
                the Line Buffer;

                Increment the Line Buffer Pointer and the file buffer
                pointer;
            }
        }
    }
    Reallocate the size of the new Line structure to N+5 bytes;
    Store the size of the new Line Structure (i.e., N+4) at offset 4 in
    Line Structure;
    Unlock the new Line Structure;
    .
    .
    .
```

The corresponding source code is located in the file named editpro.c. For EditPro, the value of MAXFILEBUFF is 0x00003FFFL bytes. This value can be made larger, if desired, to reduce the amount of disk reads required. If you look at the source code, you'll notice that when line number 32,767 (7FFFH) is reached, EditPro displays a message notifying that the file will be truncated at that point, because of Windows' scroll bar range limit of 32,767. Also, when the user exits EditPro or selects Close from the menu, we must free the memory associated with all of the Line structures.

## Painting the Screen

Painting the screen involves drawing only a portion of the text in the window. Which portion is a drawn depends on the line which the user has scrolled to and the invalidated rectangle specified by the PAINTSTRUCT structure that is filled out by the BeginPaint() function. To output the appropriate text one can use either the TextOut() or TabbedTextOut() function. EditPro uses the TabbedTextOut() function since it must expand the tabs in the file. One thing to remember about the TabbedTextOut() function is that it uses up a lot of CPU cycles whenever it is called. Therefore, you'll want to call this function as few times as possible whenever you're painting/repainting the screen. To keep track of the line to which the user has scrolled, we use a global variable, hCurrentLine, which holds the memory handle of the Line Structure of the line to which the user has scrolled. This value is set to the handle of the very first line when the file is first opened.

The variable xoffset holds a non-positive value between 0 and -32,767 which determines the text's displacement relative to the x-axis. This is used in case the window has been horizontally scrolled by the user toward the right. xoffset is changed (decreased) whenever the user scrolls the screen horizontally, and the window must be updated accordingly.

You'll notice that the text is displayed in the system font. This choice is purely arbitrary and can actually be any font that you want.

## Paging The Screen

To page up/down, whenever the user clicks the vertical scroll bar, we merely "chase pointers" either backwards or forwards from our current position in the file until we reach the line that is at the bottom of the screen or at the top of the screen. We use the client rectangle to help us find the desired line. Every time we chase a pointer to the next consecutive line, we simply add the text height of that line to our variable, CurrentLine, which initially is set to zero. When CurrentLine is greater than the client rectangle's top/bottom, then we know we've found the right line.

For horizontal paging, we merely change increment/decrement the xoffset variable and scroll the client window in the proper direction.

## Tips On Using The Caret

The topics covered thus far provide a minimum foundation for creating editors. The resulting editor merely loads and displays files. What is missing is the implemention of the system caret into the program and some basic editing operations, which are topics I cannot cover in depth in this article due to a limited amount of time. However, just to get you started on using the caret, here's a few tips:

- The caret is a shared resource. You must create the caret whenever you're window receives the input focus, and destroy the caret whenever your window loses input focus (which is signaled by the WM_SETFOCUS and WM_KILLFOCUS messages).

- It is important to hide the caret before erasing the window's background. The WM_ERASEBKGND message signals when the background must be repainted. By default (via the DefWindowProc() function) Windows paints the background. If you don't do this, what happens is that occasionally the caret will leave an "imprint" on the window. That is, you'll have two carets appearing on the screen, except only one of them will actually be the caret and the other will be just an image of the caret.

The following code is an example of how to hide the caret before the background is repainted:

```
switch( wParam)
{    .
     .
     .
     case WM_ERASEBKGND:
       // if window has input focus, hide the caret before
       // erasing...
       if( hWnd == GetFocus())
       {      HideCaret( hWnd);
              dwTemp = DefWindowProc(hWnd, Message, wParam,
                                     lParam);
              ShowCaret( hWnd);
              return dwTemp;
```

```
                 }
                 else goto DEFAULTACTION;
                case WM_PAINT:
                 .
                 .
                 .
                default:
      DEFAULTACTION:
/* For any message for which you don't specifically provide a   */
/* service routine, you should return the message to Windows    */
/* for default message processing.                               */
return DefWindowProc(hWnd, Message, wParam, lParam);
           }
           return 0L;
```

- The GetTextExtent() function must be used to calculate the new position of the caret whenever the user moves the caret.
- It's important to save the position of the caret before losing input focus in order to know where to display the caret again after regaining input focus.

## Closing Remarks

If time permits, I may write another article in the future which covers the caret and the implementation of basic editing operations.

ClickBar consists of a Dynamic Dialog pair that allows C developers for Windows 3.0/3.1 to insert a "toolbar" into their application. Microsoft, Borland, etc. developers may display a toolbar or tool palette inside their application, according to a DLG script in their .RC or .DLG file.

Borland developers may install ClickBar as a custom control, providing three custom widgets added to the Resource Workshop palette. These are three different button profiles defining 69 custom button images total. The buttons may be placed and assigned attributes identically to the intrinsic Resource Workshop widgets.

ClickBar consists of a Dynamic Dialog pair that allows C developers for Windows 3.0/3.1 to insert a "toolbar" into their application. Microsoft, Borland, etc. developers may display a toolbar or tool palette inside their application, according to a DLG script in their .RC or .DLG file.

Borland developers may install ClickBar as a custom control, providing three custom widgets added to the Resource Workshop palette. These are three different button profiles defining 69 custom button images total. The buttons may be placed and assigned attributes identically to the intrinsic Resource Workshop widgets.

Source is provided for a complete example of using the tools, including the the use of custom (owner-drawn) bitmaps for buttons.

Version 1.5 uses single-image bitmaps to reduce the DLL size, and includes source for a subclass example for inserting a toolbar.

Registration is $35 and includes a registration number and printed manual.

WynApse
PO Box 86247 (602) 863-0411
Phoenix, AZ 85050-6247 CIS: 72251,445

# *Midlife Crisis: Windows at 32*

By Pete Davis

I decided I needed to back-track this month. Last month's article was done out of a sheer need for me to know about threads and synchronization without much thought given to you, the reader. I apologize. I'm going to do my best to start from the beginning on this Win32 stuff and explain the different areas of Win32 programming. This month, in particular, I'm going to discuss the differences between the different environments, Win32 for Windows 3.1, 'Chicago', and NT). This difference is particularly important right now as far as the difference between NT and Windows 3.1 when doing Win32 programming.

One of the key differences as far as environment is that Win32 applications running under Windows 3.1 share memory with Windows 3.1 applications and Win32 applications and Win32 applications share memory with other Win32 applications. This is from a virtual point of view, if you get my meaning, and it's just what we expect from regular Windows 3.1 programs. The difference is with NT and later, 'Chicago'. Win32 programs running under NT each have their own virtual 4 GB (that's gigabytes, for those of you not used to seeing numbers of that magnitude) of RAM to work in. That means that Program Manager has its own 4 GB of RAM and File Manager has its own 4 GB of RAM, even when they're both running at the same time. Actually, when I say 4 GB, you're really limited to a mere 2 GB of that memory. The other 2 GB belongs to the Kernel. Also, keep in mind, that's 'Virtual' memory, not real memory. You're still limited by how much you have in your machine and how big your swap file is. Under NT, you have only one swap file and its the permanent swap file.

Anyway, I digress, the point is that without some sort of IPC (interprocess communication, which we'll get into in a second), NT applications wouldn't be able to see each other's memory. This makes it slightly more difficult to share memory between applications but it pays off in that one process can't damage another process' memory and cause everything to fall to pieces. Instead, under NT, the one process can hurt itself and die, but everything else goes on unaffected.

When designing Win32 applications under Windows 3.1, however, it's important to keep in mind that the applications CAN wipe each other out. If you want your applications to run under both environments, you need to be sure to keep this in mind. Under Windows 3.1, Win32 applications, like 16-bit applications, share the same memory with all the other processes running. This can be a problem when porting Win32 applications to Win32s that make assumptions about the memory.

Well, that's enough of that, I think you get the picture. Under Windows 3.1, 32-bit apps share the same memory, under Windows NT, they don't. Clear?

Ok, so what if you want to share memory? Well, DDE is still there, but I wouldn't recommend it for most situations. It's just too much of a pain for most applications. The old method of using GlobalAlloc and then passing the handle in lParam of a message and the other application using GlobalLock to access it no longer works because that 'Global' memory isn't so global anymore, it's not shared between applications (except in Win32s, but we don't want to make that assumption, right?). In the Windows NT environment, there are memory-mapped files. These are an excellent method of IPC and the good news is that Microsoft is working on implementing it for Win32s (that's 32-bit Windows for Windows 3.1, as opposed to Win32 which is for NT and Win32c for 'Chicago'). This means there will be a simple, common method for sharing data between applications.

I won't go too in-depth into it this month, but memory-mapped files are essentially files that are treated as if they're regular memory, except a sort of temp file is created to hold the data. The way it works is that you have a file that you treat as if it's memory. You simply have a pointer that you can point and move anywhere in the file as if it were a regular pointer to memory and Win32 picks up the tab on the file I/O work. It's basically just another type of virtual memory. All the other program has to know about is the file mapping object name. There are other uses for memory-mapped files which I'll cover in a later column, likely one devoted to memory-mapped files.

## Preemptive Multitasking vs. Non-Preemptive Multitasking

Windows NT (and Windows 4.0, when it gets here) is a preemptive multitasking operating system. You've probably seen those buzzwords mentioned in every review or discussion of NT. What does this mean exactly? Preemptive multitasking is where the CPU decides how much time your program has to run. After that time is up, your program is stopped, the register contents are saved and another program is given control. And then another and then another until it gets back to the first. Then its registers are reset and it is sent on its way for its particular time slice.

Windows 3.x use a non-preemptive multitasking system where each program is responsible for giving the other programs in the system time to run. This cooperation takes place in the message queue where messages are passed around the system from one program to another until it finds its way to the program that needs it. In Windows 3.x it's simple to write a program that doesn't release the CPU to other applications. Just try writing an infinite loop and you've locked up the entire system.

Under Windows NT, the message queue is still used, but the system doesn't rely on it to provide CPU time to other programs. If your program has an infinite loop, then your program will lock up. The rest of the system will continue running with no effect from your program.

I should mention that Windows 3.x is not an entirely non-preemptive multitasking system. There are a couple parts that are, or can be, preemptive. The DOS box, for example, is preempted by Windows. The DOS box receives its time slice to run and that's it. Then Windows gets some time for a while and then it gives another time slice to the DOS box. Another place than can be preemptive is VxDs or .386 drivers. The reason is that these run at what is called Ring 0. Ring 0 is where the low-level system control stuff takes place in 386, 486 and now Pentium chips. (There are four rings, 0-3. Rings 0 and 3 are the only ones used by Windows). Actually, this area isn't exactly preemptively multi-tasked but is available for it. The software running at Ring 0 has complete control over the system, so it can really do whatever it wants.

That's all for this month, I'm afraid. I'm really sorry if my articles seem a bit anemic for the next few months. I'm afraid I have little choice in that. We'll see how much time I actually have over the next few months, but this book is really going to take a lot of work.

# *Software Review: WDASM Windows Disassembler*

By Pete Davis

WDASM is the only Windows disassembler I know of that's available for shareware. The product is amazing and well worth it's mere $10 registration fee. The version I have been using is 1.2, which is the unregistered version. I am awaiting version 1.6 which is on it's way.

WDASM is available from many sources, including the CompuServe WINSDK forum, WSMR-SIMTEL20.ARMY.MIL in the PD:<MSDOS.WINDOWS> directory, and from FTP.CICA.INDIANA.EDU in the pub/pc/win3/programr directory.

When you run WDASM you get a window with a small and simple menu. The menu structure is:

| File | Edit | View | Help |
|---|---|---|---|
| ---- | ---- | ---- | ---- |
| Open | Set Bytes | Segment | Index |
| Save Text As | | Goto | About |
| ---- | | Address Offset | |
| Exit | | Far Call Names | |

WDASM will disassemble both .EXEs and .DLLs. As soon as you open an executable, the WDASM window shows the disassembly of Segment 1 of the executable. To change to different segments, simply go to View and select the segment you want.

Set Bytes allows you to set a range of bytes to data, instructions, labeled instructions.

Goto, of course, takes you to a certain address. Address Offset is a nice feature. Normally, the only address offsets that show up in the code are the labels. All labels are in the form:

LxxxxH: where xxxx is the offset.

Far call names toggles between labeling far calls and showing the actual address of the call.

When you're done you can use Save Text As to save the assembly language code generated by WDASM to a .ASM file.

There are some things I'd like to see in WDASM and, in fact, after talking to Eric, it appears some of them are there (in the registered version). These include some minor bug fixes and some expansion of the program. I'd like to see a little commenting in the code about what's going on. Sometimes it's a simple operation to find out what some code is doing. It would be nice if the disassembler did some of those for you.

Also, I'd like to be able to do some editing in WDASM. For example, if I figure out what a certain routine does, I'd like to name it in WDASM and have it go through and correct all references to that label.

I'd like support for the PE file structure, which, actually, when I suggested it to Eric, he was more than happy to get some information on PE file structures from me and plans to upgrade the software to handle them.

Other upgrades which I'm aware of that are in the works (or done?) include support for 386/486 instructions and support for floating point instructions.

I have a wish list of things to be added and I have passed a lot of them to Eric. This is where Eric's product sticks out the most. I've talked to other shareware authors about improvements in their products. Few have been as receptive to new ideas as Eric. Eric does his product for the end-user, not himself. That's really not as common as you might think. A lot of shareware authors ask for your comments and when you send them in, just ignore them. Eric really wants to do a product that will be useful for his users. He's already done that.

I can't recommend this program highly enough. If you do Windows disassembly, consider this program. For the price, it's an absolute bargain. And if you like it, PLEASE REGISTER it. It's only $10, which is inexpensive, even for shareware. It's worth every penny.

Happy disassembling.

[Ed. Note: Just prior to the release of this issue I received Version 1.6 of the disassembler. Not enough time to really use it a lot or to rewrite the review. I will only say that 1.6 has some major improvements. I also know Eric is in the process of making more. Get this disassembler and try it out.]

# *Hypertext, Sex and Winhelp*

By Loewy Ron

Hi, this article is about hypertext, help systems and winhelp. If you ask yourself what has this kind of article to do in a programming magazine, and tell yourself (and everyone around you) that you are a programmer and not a technical writer, I can only tell you of a sentence I heard at a recent conference I attended, from a Rep. of a firm that markets a database development tool. The guy told us that in user surveys they learned that people that evaluated their product, gave the quality of the documentation the same weight they gave to the functionality of the product. (Hearing this stuff, and knowing I'm not as good as I should be at writing documentation, I decided I had to stop while I'm at the top - so this is the end of this article, bye. Well, no - this was just a joke, we will continue from now ..).

An important part of your application's documentation is the on-line help. One of the advantages of programming for Windows, is that Windows includes a built-in help engine, with a (well?) documented interface. This engine is both easy for the users to use, standard among applications, and yet is powerful and extendible.

This article will talk a bit about hypertext systems in general, and will try to focus on the Windows implementation.

Hypertext is a term used to describe large amounts of textual data, that have embedded references to other textual elements of the database. These other elements have some connection to the text element that is currently read by the user, and might interest him.
The difference of hypertext systems from traditional documentation is that reading a hypertext document, the reader (user) can read in a non linear order, and jump around the database according to his will (and the references that were created in the document by the system's designer).

We usually define a piece of textual information that can be regarded as a single unit - a Topic. Our help "database" is a collection of Topics, that most (if not all) of them, have references (called Links) to other Topics, that have some connection to them.

You have probably used some hypertext systems, such as the on-line help that came with your development package/language/compiler. (if you still use GW-BASIC as your main development tool - you are advised to upgrade, or you might find some of the other articles in this journal hard to follow).

In order to incorporate help (using winhelp) into your application, you (roughly) have to go through the following steps :

1. Design the layout of your hypertext help database.

2. Create the winhelp source files, and compile them to create the .HLP file(s) for use with the winhelp application.

3. Create a help menu/button/toolbar or other interface element from your application to the winhelp database, and code the winhelp API calls to activate that database from your application.

4. Optionally - If you want to create context-sensitive help (help that is activated when a user presses the F1 key for example, and receives help about the operation/options that he is currently doing/having), code the key interception and calls to the winhelp database.

Step 1 is really something that should be left to people that know more about user interfaces, usability studies and these sort of things than me. I have included some references that might be of interest at the end of this article.

Steps 3 and 4 will be discussed in a future article (If the editor will want me to write any more after some angry readers will e-mail him some threats, or knock on his door at the middle of the night).

In order to create .HLP files - that contain your help database, you have to create source files for Microsoft's help compiler. There are actually 2 help compilers - HC30.EXE that creates help topics in Windows 3.0 format, and HC31.EXE that creates help topics in Windows 3.1 format. While HC31 help databases can be more sophisticated than HC30 created databases, you will not be able to use them on Windows 3.0 platforms, while winhelp that came with Windows 3.1 can display help from .HLP files in both formats.

The input to the help compiler is a project description (.HPJ) file, and a set of Rich Text Format (.RTF) files, with special code to describe Topics, Links, Search keywords etc..

RTF is a document description standard that can describe (any?) word-processor file using only 7 bit characters. Unfortunately RTF is a language that is very hard to read and maintain manually. You would probably want to use a word processor that can output RTF files, or a tool that can translate plain readable ascii files into RTF output.
If you have Word for Windows, several help authoring templates, styles and macros are available in the commercial and shareware market, that can help in the creation of help databases. At the end of this article you will find a list of the help authoring packages I am aware of, I have not tried or used all of them, I'm just trying to bring them to your attention.

In the RTF file, use page breaks between Topics. Your Topics can be longer than one page, but when you use a page break - you tell the Help Compiler that this is the end of the text that belongs to this Topic. For each Topic include a # and $ footnotes to describe its name, and reference, so that other topics will be able to include Links to it. Links are included using a text with double underline attribute, followed by a hidden Topic reference name. I will give a RTF example that will help you understand the mechanism.

We will define 2 Topics - TOP1 and TOP2 and create Links between them. The following code example is part of a RTF file.

```
{\f2
 #{\footnote \pard\plain \fs20 # TOP1}
 ${\footnote \pard\plain \fs20 $ TOP1}
}
\pard{This is the first paragraph of Topic 1, with no Link here}
\pard{This is the 2nd. paragraph of topic 1, we will include a
{\f2\uldb Link}{\v\f2 TOP2} here to Topic 2}
\page
{\f2
 #{\footnote \pard\plain \fs20 # TOP2}
 ${\footnote \pard\plain \fs20 $ TOP2}
}
\pard{This is Topic 2, Click {\f2\uldb Here}{\v\f2 TOP1} to reach
topic 1 }
\page
```

As can be seen - Each topic starts with a # and $ footnotes, and ends with a page break (\page). Whenever we want to create a Link, we use the \uldb (double underline) attribute with the text we want to display, and follow it with a hidden (\v) text with the reference to the Topic it is linked to.

It should be explained at this point - that this code is not enough to create a help database that contains only these 2 Topics, it is, however, beyond the scope of this article to describe all the RTF commands needed to create a help document. I have, however, included the source to a simple help database with only 3 Topics, and Links between them. You will have to refer to the RTF specifications (available from Microsoft, or - if you have the MSDN CD-ROM, you can find it there), and the help compiler documentation that comes with the SDK or with your compiler.

The next file that should be created for the help compiler is the project file. The following code is an example of such a file :

```
[OPTIONS]
INDEX=TOP1
COMPRESS=TRUE
TITLE=Demo Help Project

[FILES]
DEMOPROJ.RTF

[MAP]

[BITMAPS]
```

This is a file that have a structure that resembles a .INI file, with sections and options in it. The [OPTIONS] section include general options that are used by the help compiler during the .HLP file generation. In the example above our Topic 1 is defined as the index topic - the one that winhelp will display first when it loads the help file, the compress entry tells the help compiler to compress the help text, and the title entry instructs winhelp what to display in the title of the help window. It is important to notice that the .WPJ file syntax is different between HC30 and HC31. The example above is good for HC30, and in HC31 the index entry should be replaced by a contents entry to achieve the same results.

The [FILES] section lists all the RTF source files used to build the help database. There are additional sections and entries in section that are used to create and set options to the help database. It is advised that you refer to the Microsoft SDK documentation, or the documentation that came with your compiler for a complete discussion of all the features and options of the help project file.

Summary
-------

O.k, this was just an introduction that touched the surface of what it means to create help databases for use with winhelp, and this should probably get you started. I might add additional articles on this subject in the future. Good luck, and have fun helping others.

If you wonder about the title of the article - hey, this is my first one, and I covered 2 of the 3 subjects - that is pretty good, don't you think?

References :
------------

1. Microsoft Press - Microsoft Windows 3.1 Programming Tools, Chapter 3 and Appendix B.

2. Borland Pascal With Objects 7.0 - Tools and Utilities Guide, Chapter 6 and Appendix A.

3. Windows Help Authoring Guide - A Microsoft publication available on the internet as WHAG.ZIP in CICA.

4. RTF Specifications - I have mine from the Microsoft MSDN CD-ROM.

Help Authoring Tools :
----------------------

This is a list of tools I know that exist. I have not used all of them, nor do I know if they all work. I only included this list in order to help you get a jump start in creating help databases. All of the opinions I express - are mine, and mine only.

1. Microsoft Windows Help Authoring Tools - a combination of a help project editor and winWord templates and macros. Available on the internet in CICA as WHAT.ZIP.

2. SYNTAX.ZIP - A tool to convert WP documents to windows help. Available in the misc directory of CICA on the internet.
3. HELP.ZIP - A tool to convert OS/2 IPF files to RTF documents. Available in the nt directory of CICA on the internet.

4. HAG.ZIP, WHAG.ZIP - Windows help authoring guide in winword and
   .HLP formats. Available in the programr directory of CICA.

5. WFWHLP.ZIP - Info on Constructing Windows Help Files.
   Available on the program directory of CICA, also in CIS, WINSDK
forum, Lib. #16.

6. HWAB21.ZIP - Help Writer's Assistant for Windows (This is cute).
Available on the util directory of CICA.

7. QDHELP.ZIP, QDTOOL.ZIP - Use DOS Editors to Create WinHelp Files.
This is a really nice tool. Available on the util directory of
   CICA. Also on CIS WINSDK Forum, Lib 16.

8. DRHELPEV.ZIP - Macro to translate Word files to WinHelp files.
   Available on the winword directory of CICA.

9. HLPDK30.ZIP, HLPDK40.ZIP - My own Help Development Kit. I'm biased so I
will not say any further word. Available on SIMTEL hypertext directory,
GARBO programming directory, Compuserve WINSDK forum, and an undefined
place (currently in uploads) on CICA.

10. Doc-To-Help - A commercial tool. A press release can be found in
D2HPRS.TXT and D2HNEW.ZIP in library 3 of the WINSDK forum on COMPUSERVE.

11. HELP.DOT - WFW 2.0 Template for Creating WINHELP RTF Files -
    Library 3, WINSDK Forum, CIS.

12. RoboHELP - A Commercial Tool (I have heard some nice things about
this one). HELPTO.TXT on CIS WINSDK forum (Lib 3) has a reference to it.

13. BOOKHL.ZIP - Developing Online Help for Windows - The Book.
    WINSDK forum, Lib 16, CIS.

14. CH102.ZIP - CreateHelp! HLP authoring tool.
    WINSDK forum, Lib 16, CIS.

15. HCMAC1.ARC - HC1.DOT template and support files to build help.
    WINSDK forum, Lib 16, CIS.

There are more tools and references in CIS WINSDK Forum, LIB 16, as
well as other places.

# *Enhancing the WINSTUB Module*

By Rodney Brown

When I was reading over the first issue of WPJ, I came upon an article that was talking about the possibilities of the WINSTUB module. This got me thinking and I started working on enhancing WINSTUB. WINSTUB is a DOS program that is placed at the front of your Windows application. When you try to load a Windows program from DOS, it is the culprit that gives you that nasty little message "This program requires Microsoft Windows," then drops you back to DOS to pay for your sin<g>.

The first WINSTUB program I designed automatically loaded up Windows if the program was started from DOS. I then decided to change it a little bit because some users may not like that. The second WINSTUB module asks the user whether they want to load Microsoft Windows or exit to DOS. This allows the user to back out of the program if they made a mistake in typing the program name, etc.. The WINSTUB program was written in Turbo C++ for DOS. The source code and project file are provided in the WPJ archive, but the C++ code is discussed below for those with other C/C++ compilers.

```
// Enhanced WINSTUB.EXE for Windows applications
```

```
#include <process.h>
```

The process.h header file contains the information for the spawn command, the spawn command allows you to call an external program.

```
#include <stddef.h>
```

This header file is used solely for the definition of the NULL statement.

```
#include <iostream.h>
```

This is the C++ version of stdio.h

```
char answer;
int returncode;
```

These two statements initialize an answer variable of type char, and a returncode variable of type int.

```
int main(void)
{
  cerr << "\nThis program requires Microsoft Windows to run.";
  cerr << "\nWould you like me to start Microsoft Windows for you? ";
```

The cerr << "text" command is the C++ equivalent of sending text to stderr.

```
  cin >> answer;
```

The cin >> variable command is the C++ equivalent of retrieving data from stdin.

```
 if (answer == 'y' || answer == 'Y')
     {
        returncode = spawnlp(P_WAIT,"win.com","","myprog",NULL);
```

The statement above tries to call win.com (Windows). If the call is not successful, the spawn command returns a negative integer number. The "l" in spawnlp signifies that the number of parameters sent to called program is known. The "p" in spawnlp tells spawn to search the DOS path for the program being called.

```
        if (returncode<0)
           {
              cerr << "\n\nError, could not start Microsoft Windows.";
              cerr << "\nExiting back to DOS...";
           }
     }
  else
     cerr << "\n\nExiting back to DOS...";

  return returncode;
}
```

The purpose of the rest of the program should be obvious, if the user answered yes and the spanwnlp call was not successful, an error message is displayed and the program drops back to DOS. If the user answered no, the program exits to DOS.

Once you have written and compiled your DOS WINSTUB program, you need to return to window and modify your .def file to put everything together. A sample .def fil
After reading the last 4 WPJ's, you should know what most of the above code does. The main parameter we are concerned with is the STUB parameter. In Turbo C++ for Windows (and I assume Borland C++), the STUB parameter is not required unless you are using a WINSTUB module other than Borland's. Turbo C++ for Windows also does not require the EXPORT parameter if you are using ObjectWindows.

There are other possibilities for WINSTUB, the program above was just a simple enhancement of WINSTUB. If you thought the comments of the source code above was overkill, I'm sorry, but I just wanted to explain the program to people we looks like this:

```
NAME              MYPROG
DESCRIPTION       'Description of program goes here...'
EXETYPE           WINDOWS
STUB              'WINSTUB.EXE'
CODE              PRELOAD MOVEABLE
DATA              PRELOAD MOVEABLE MULTIPLE
HEAPSIZE          4096
STACKSIZE         5120
```

## About the author

Rodney M. Brown is a civilian employee at Charleston Air Force Base, South

Carolina. His job involves operating and maintaining minicomputers and POS Systems, programming and repairing microcomputers. He also gives computer lessons to fellow employees.

He can be reached through the following services:

GEnie: R.BROWN141

CompuServe: 72163,2165

Internet: 72163.2165@compuserve.com

If you are connected to more than one of these services, communication through CompuServe is preferred.

## *Microsoft Developer's Network*

By Dave Campbell

How would you like to have all the latest information about Windows programming, OLE, ODBC, etc. at your fingertips? This would include sample code from the people that wrote the compiler and defined the interface, and books by leaders in the industry. So far I'm talking about 2-layers of documentation deep on my desk.

That's what I usually get into when I get rolling on my article, or get deep into a development sequence. Sometimes it gets downright embarrassing.

There is a solution, though. If you haven't heard about the Microsoft Developer's Network yet, you're missing out on a valuable resource. I have had the pre-release CD since December, but didn't get a CD-ROM drive until this month. This is the scenario: I got into Windows, then Word, and opened up my article, and my source code. Then back to Resource Manager, and opened the CD. Now for the rest of the article, I worked back and forth between the two Windows applications, cutting and pasting, and rewording, and basically saving me a ton of time.

The Developer's Network is a subscription program in which Microsoft sends a CD out quarterly which uses a great user interface to search out information. The pre-release CD contains technical information and example code to use in writing applications for C/C++, DDE, OLE, using the Clipboard. NT and VB 2.0 sample code, VB 2.0 Professional documentation, Excel SDK, MAPI, ODBC, and RTF specifications. Additionally you'll find Charles Petzold's Programming Windows 3.1 book in its entirety, including code with hyperlinks out to the SDK. There's a few years worth of Microsoft Systems Journal magazines, and Ray Duncan's "Advanced MS-DOS Programming" book in its entirety.   When you open up a section of the CD for inspection, you can view the code, run the executable, or cut and paste directly into your application from the CD.

I think you get the idea.

The installation was extremely smooth...all contained on the CD itself. My CD-ROM drive normally comes alive as drive I:, so I just ran I:\SETUP.EXE. It is taking up about 10M of my hard disk because I let it do the most it wanted, for speed of access later. Once it was finished, the whole thing was available for use.

The cost of the program is $200/yr now that the pre-release program is over. Petzold's book alone is $50 if you buy high-dollar retail, so that covered the cost of the first one for us in the pre-release. I'm sure I won't be disappointed with the remainder of them.

Of course, you'll need a CD-ROM drive to play the CDs, and that isn't cheap yet. But, if you know someone with a Gateway 2000 customer ID, you can buy a Sony drive, which is AX, CDI, multisession, and Kodak compatible, drive, cables, and interface card for $225 plus $5 shipping. I highly recommend finding someone that can order this for you. I got mine in 5 days, and it took about 20 minutes to install. It plays audio and digital

CDs, and works great with the Developer's network.

If you have any questions, feel free to send me e-mail:

Dave Campbell
WynApse PO Box 86247 Phoenix, AZ 85080-6247 (602)863-0411
wynapse@indirect.com
CIS: 72251,445
Phoenix ACM BBS (602) 970-0474 - WynApse SoftWare forum

## Getting in Touch with Us.

Getting in touch with us:

Internet and Bitnet:

HJ647C at GWUVM.GWU.EDU -or- HJ647C at GWUVM.BITNET (Pete)

GEnie: P.DAVIS5 (Pete)

CompuServe: 71141,2071 (Mike) 71644,3570 (Pete)

Delphi: PeteDavis

America Online: PeteDavis

WPJ BBS (703) 503-3021 (Mike and Pete)

You can also send paper mail to:

Windows Programmer's Journal
9436 Mirror Pond Drive
Fairfax, VA 22032
U.S.A.

# The Last Page

by Mike Wallace

Some background: Taoism is a very old philosophy that originated in China, and one of its major writers was a man named Chuang-tse. This month's column starts off with a story from his writings:

> Hui-tse said to Chuang-tse, "I have a large tree which no carpenter can cut into lumber. Its branches and trunk are crooked and tough, covered with bumps and depressions. No builder would turn his head to look at it. Your teachings are the same - useless, without value. Therefore, no one pays attention to them."
>
> "As you know," Chuang-tse replied, "a cat is very skilled at capturing its prey. Crouching low, it can leap in any direction, pursuing whatever it is after. But when its attention is focued on such things, it can be easily caught with a net. On the other hand, a huge yak is not easily caught or overcome. It stands like a stone, or a cloud in the sky. But for all its strength, it cannot catch a mouse.
>
> "You complain that your tree is not valuable as lumber. But you could make use of the shade it provides, rest under its sheltering branches, and stroll beneath it, admiring its character and appearance. Since it would not be endangered by an axe, what could threaten its existence? It is useless to you only because you want to make it into something else and do not want to use it in its proper way."

Why am I writing about Taoism? Well, I'm not, really. I was reminded of this story a couple of weeks ago after reading a review of IBM's OS/2 2.1. The focus of the article seemed to be that OS/2 could run Windows programs almost as well as Windows. Sounded like the reviewer wanted to make OS/2 something it's not: Windows. Thus, the story. Now, I'm not here to harp on that useless operating system OS/2, but if I want to run Windows, I'm not going to buy a 486 with 12MB of RAM and a 100MB hard drive to support OS/2. I've seen Windows run fine on a 386 with 4MB and a much smaller hard drive. My computer could run OS/2, but I installed Windows NT on it, and it runs Win3.1 programs without a problem. Windows NT has about the same memory requirement as OS/2 but needs less space on your hard drive.

Windows and OS/2 are different operating systems, and I think they were each designed with a different kind of end user in mind. If you want to run Windows, why buy OS/2? OS/2 should have a better selling point than "it runs Windows apps almost as well as Windows." Anyone buying OS/2 based on this promise will only be disappointed, I think. An operating system should be able to stand on what makes it unique, or it will end up being nothing more than a cheap imitation.

Hey, enough ranting and raving. This issue is already late enough. Talk to you next month.