Microsoft Windows Programmer FAQ

Frequently Asked Questions

Note: *revision dates for each section are shown next to the section names on each index page!*

Credits

The author may be contacted by the following means:

Internet:	tomh@wes.on.ca
UUCP:	uunet!watserv1!wes!tomh
CompuServe:	>INTERNET: tomh@wes.on.ca
Mail:	Tom Haapanen Waterloo Engineering Software 22 King St.S., suite 302 Waterloo, Ont. N2J 1N8, Canada

The WinHelp conversion macros were created by Roger Hadgraft, senior lecturer in Civil Engineering at Monash University, Clayton, Victoria, Australia.

Internet:	roger.hadgraft@eng.monash.edu.au
UUCP:	uunet!eng.monash.edu.au!roger.hadgraft
CompuServe:	>INTERNET: roger.hadgraft@eng.monash.edu.au

Microsoft Windows

Windows 1.0

Microsoft first began development of the Interface Manager (subsequently renamed Microsoft Windows) in September 1981. Although the first prototypes used Multiplan and Word-like menus at the bottom of the screen, the interface was changed in 1982 to use pull-down menus and dialogs, as used on the Xerox Star.

Microsoft finally announced Windows in November 1983, with pressure from just-released VisiOn and impending TopView. This was after the release of the Apple Lisa (but prior to the Macintosh), and before Digital Research announced GEM, another competing graphical environment. Windows promised an easy-to-use graphical interface, device-independent graphics and multitasking support. The development was delayed several times, however, and the first version hit the store shelves (after 55 programmer-years of development!) in November 1985. The selection of applications was sparse, however, and Windows sales were modest,

The following were the major features of Windows 1.0:

- Graphical user interface with drop-down menus, tiled windows and mouse support
- Device-independent screen and printer graphics
- C˜o-operative multitasking of Windows applications

Windows 2.0

Windows 2.0, introduced in the fall of 1987, provided significant useability improvements to Windows. With the addition of icons and overlapping windows, Windows became a viable environment for development of major applications (such as Excel, Word for Windows, Corel Draw!, Ami, PageMaker and Micrografx Designer), and the sales were spurred by the runtime (Single Application Environment) versions supplied by the independent software vendors. When Windows/386 (see next section) was released, Microsoft renamed Windows to Windows/286 for consistency.

The following are the major changes from earlier versions of Windows:

- Overlapping windows
- PIF files for DOS applications

Windows/386

In late 1987 Microsoft released Windows/386. While it was functionally equivalent to its sibling, Windows/286, in running Windows applications, it provided the capability to run multiple DOS applications simultaneously in the extended memory.

The following are the major changes from earlier versions of Windows:

- Multiple DOS virtual machines with pre-emptive multitasking

Windows 3.0

Microsoft Windows 3.0, released in May, 1990, was a complete overhaul of the Windows environment. With the capability to address memory beyond 640K and a much more powerful user interface, independent software vendors started developing Windows applications with vigour. The powerful new applications helped Microsoft sell more than 10 million copies of Windows, making it the best-selling graphical user interface in the history of computing.

The following are the major changes from earlier versions of Windows:

- Standard (286) mode, with large memory support
- 386 Enhanced mode, with large memory and multiple pre-emptive DOS session support
- No runtime versions available
- Program Manager and File Manager added
- Network support
- Support for more than 16 colors
- API support for combo boxes, hierarchical menus and private .ini files

Windows 3.1

Microsoft Windows 3.1, released in April, 1992 provides significant improvements to Windows 3.0. In its first two months on the market, it sold over 3 million copies, including upgrades from Windows 3.0.

The following are the major changes from Windows 3.0:

- No Real (8086) mode support
- TrueType scalable font support
- Multimedia capability
- Object Linking and Embedding (OLE)
- Application reboot capability
- Better inter-application protection and better error diagnostics
- API multimedia and networking support
- Source-level API compatability with Windows NT

Windows NT

Microsoft Windows NT, scheduled for release at the end of 1992, is Microsoft's platform of choice for high-end systems. It is intended for use in network servers, workstations and software development machines; it will *not* replace Windows for DOS. While Windows NT's user interface is very similar to that of Windows 3.1, it is based on an entirely new operating system kernel.

The following are the major changes from Windows 3.1:

- Based on a new microkernel design
- Portable architecture for Intel x86, MIPS R4000 and DEC Alpha processors
- 32-bit addressing for access to up to 4 GB of memory
- Fully protected applications with virtualized hardware access
- Installable APIs for Win32, Win16, MS-DOS, POSIX and OS/2
- Installable file systems, including FAT, HPFS and NTFS
- Built-in networking (LAN Manager and TCP/IP) with remote procedure calls (RPCs)
- Symmetric multiprocessor support
- Security designed in from start, to be initially C2 certified, with a B-level kernel design
- API support for unsynchronized message queues, advanced interprocess communication, registration databases, Bezier curves and graphics transformations.

Although Windows NT has not yet been released, the following is generally accepted as the minimum platform for use with the retail release of Windows NT:

- 33 MHz 386 processor
- 8 MB memory
- 100 MB hard disk
- VGA graphics

As of July 1992, Windows NT is available as a pre-release SDK (Software Development Kit) from Microsoft at the cost of $69 (or $399 for the pre-release SDK plus full printed documentation). This release is supplied on CD-ROM only, and contains the Windows NT operating system as well as all the necessary 32-bit development tools (including a 32-bit C++ compiler and all documentation online on the CD-ROM). The purchasers of this SDK will also receive free updates to Windows NT up to and including final release.

The pre-release Windows NT SDK requires 12 MB of RAM and is **not** suitable for evaluating the Windows NT environment. It is intended strictly for software development! Contact your national Microsoft subsidiary (or Microsoft itself in Redmond, WA) for ordering information.

Win32s for Windows 3.1

Win32s is a set of libraries for Windows 3.1, which enable users to run most Windows NT 32-bit applications on Windows 3.1, without the extensive hardware requirements of Windows NT. The Win32s interface will likely replace the older Windows-32 interface used by current 32-bit Windows applications such as Mathematica.

Win32s has not yet been released, and Microsoft has not released minimum system requirements for using Win32s applications.

Windows for Pen Computing

Microsoft developed Windows for Pen Computing for use on pen-based systems. In most aspects, it is basically equivalent to Windows 3.1 with extensions for pen support. These extensions include the useof a pen as a pointing device as well as handwriting recognition and conversion. Pen Windows first shipped in April, 1992.

Multimedia Windows

The term Multimedia Windows describes a package with Windows 3.0 and the Multimedia Extensions. These extensions are included in Windows 3.1, and thus Multimedia Windows is no longer sold as a separate product.

WinOS2

WinOS2 is the Windows component of IBM's OS/2 2.0. It is based partially on Windows 3.0 and partially on 3.1. While it runs a majority of the commercial Windows applications, it is not covered by this document.

Internet and Usenet

Usenet

If you received this FAQ from somewhere other than Usenet or Internet, you may not be familiar with Usenet. Basically, Usenet is a loose collection of over 100,000 computers which exchange mail and news. The network is unstructured and highly distributed; most communication is either by TCP/IP over high-speed connections, or UUCP over public telephone lines. Internet is a (almost proper) subset of Usenet, consisting of somewhere between 50,000 and 100,000 computers connected by high-speed TCP/IP network connections.

Usenet *news* is a software system where a person can post an article to a selected newsgroup, and have every other news reader be able to read it. There are over 1000 newsgroups (including the *alt* groups), and daily volume of news is approaching 40 MB.

While most Usenet systems are Unix-based, it is not a requirement. If you have an Internet or UUCP connection, ask your system administrator whether you have Usenet news available. Some of the most common newsreading software packages are *readnews*, *rn*, *trn*, *nn* and *notes*.

Usenet Windows newsgroups

There are a total of eight Usenet newsgroups dealing with Microsoft Windows:

- **comp.os.ms-windows.advocacy**
  This group is intended for adversarial discussions, arguments and comparisons to other computers and operating systems.
- **comp.os.ms-windows.announce**
  This is a low-volume moderated group with only Windows-related announcements and with no discussion.
- **comp.os.ms-windows.apps**
  This group contains discussions, questions, and comments about the selection and use of Windows applications.
- **comp.os.ms-windows.setup**
  This group is meant for questions and discussions about Windows setup process, driver availability and selection, and hardware compatability and selection.
- **comp.os.ms-windows.misc**
  All other discussions about Windows should be in this group.
- **comp.os.ms-windows.programmer.tools**
  This group is intended for discussions about the selection and use of tools for Windows software development.
- **comp.os.ms-windows.programmer.win32**
  All discussions about the Win32 applications programming interface (used in Windows NT and Win32s) and the Windows NT SDK belong in this group..
- **comp.os.ms-windows.programmer.misc**
  This group is for all other discussions about Windows software development.

The following groups have been replaced by those shown above:

- **comp.windows.ms**
  This group was for discussions about Microsoft Windows.
- **comp.windows.ms.programmer**
  This group was for discussions about programming for Microsoft Windows.

The following groups may also be of interest:

- **comp.os.msdos.programmer**
  This groups contains general MS-DOS programming questions. Some, especially those concerning compiler selection, may be of interest to Windows programmers.
- **bit.listserv.win3-l**
  This group is a two-way gateway of the BITNET *WIN3-L* mailing list.

The following groups are **not** for Microsoft Windows!

- **comp.windows.misc**
  This group is for miscellaneous discussions about windowing systems in general.
- **comp.windows.news**
  This group is for dicussions about the Sun Microsystems NeWS windowing system.

In general, these newsgroups are only available to computers connected to Usenet or Internet; they are not gatewayed into BITNET, CompuServe, Prodigy or other services. Some FidoNet BBS systems, however, do carry selected Usenet newsgroups.

Alternatives to Usenet

If you are unable to find a connection to the *Internet* (that procedure can not be easily defined, as the *Internet* does not have any sort of a formal structure), there are several alternatives available for finding more information about Windows, and for locating Windows software and drivers.

BITNET users (as well as any other with an electronic mail connection to Internet) can subscribe to **WIN3-L** *(win3-l@uicvm.bitnet),* a mailing list dedicated to Windows discussions.   This mailing list is similar in content to the **comp.os.ms-windows.misc** newsgroup; no programmer mailing list exists on *BITNET*.

If you live in North AMerica (or in one of selected Western European countries), you can subscribe to *CompuServe*, a commercial service.   *CompuServe* has extensive Windows-oriented discussions and a fairly good selection of free software.   Although the level of discussion is often less technical, it is much more structured than the *Internet.*   *CompuServe* also has numerous vendor-supported forums, including ones organized by Microsoft for Windows and Windows NT.

Many *FidoNet*-based BBS systems also carry the *Internet* Windows newsgroups.   Consult a local BBS listing to find your nearest *FidoNet* BBS.

Freeware and shareware by ftp

While CompuServe (which has a lot of software) and your local BBS may have large selections, the Internet provides an immense resource for all PC users.   The key program to access this software is called ftp (File Transfer Protocol), and it's useable from most Internet system, but is not usable through UUCP links.

If you do have ftp available to you, follow the example below to connect to ftp.cica.indiana.edu (do not type in the // comments):

```
$ ftp ftp.cica.indiana.edu          // make connection
Connected to ...                    // cica responds
Userid (user@cica): ftp             // enter "ftp" as userid
Password: real_userid@site          // enter your own userid
ftp> tenex                          // for binary transfers
ftp> cd /pub/pc/win3                // where the goodies are
ftp> ls -l                          // list the directory
ftp> get ls-ltR                     // get the current index
ftp> quit                           // we're done!
$ _
```

Of course, you can get multiple files at a time   read the ftp manual page for more information.

Remember that **shareware is not free**: register the software you use to encourage the development of more low-cost software.

Popular Internet ftp sites

The following ftp sites provide significant amounts of software of interest to Windows users:

- **ftp.cica.indiana.edu *(###.###.###.###)***
  Directory */pub/pc/win3* contains one of the largest selections of Windows software and device drivers anywhere. *Please do not access ftp.cica.indiana.edu between 8am and 6pm EST to prevent overloading the system.*
- **simtel10.army.mil *(###.###.###.###)***
  Directories */msdos1, /msdos2* and */msdos3* contains a very large selection of MS-DOS (and some Windows) software.
- **wuarchive.wustl.edu *(###.###.###.###)***
  Directory */mirrors/win3* contains a copy of the *cica* Windows archives, and directory */mirrors/msdos* contains a copy of the *simtel10* MS-DOS archive..
- **garbo.uwasa.fi *(###.###.###.###)***
  Directory /### contains a majority of the *cica* Windows archives. *Note that garbo.uwasa.fi is located in Finland, and North American users should avoid congesting transatlantic Internet links by ftping from this site.*

If your ftp program complains about an unknown site, you can substitute the numeric address (shown after each site name above) for the name in the ftp command.

Ftp by email

There are several sites that will perform general FTP retrievals for you in response to a similar mail query, although it appears that the *info-server@cs.net* server is permanently out of order.

In general, please be considerate, and don't over-use these services. If people start using them to retrieve megabytes and megabytes of GIF or WAV files, they will probably disappear. Also, keep in mind that your system may be linked to the net using a long-distance UUCP connection, and your sysadmin may not be happy about large mail files using up modem time and filling overloaded spool directories.

- **bitftp@pucc.bitnet**

For information on this one (available only to BITNET sites) send it the message:

```
help
```

- **ftpmail@decwrl.dec.com**

For information on this server, available to all Internet sites, send it a mail message with a body containing simply:

```
help
```

- **mailserv@garbo.uwasa.fi**

One final choice is to use the *garbo.uwasa.fi* server, which lets you access the *garbo.uwasa.fi* archive (which contains most of the *cica* files). For instructions, send it a mail message with *"Subject: garbo-request"* and a single line of text "send help" to

```
send help
```

Please do not use this service if you are in North America!

Software Development Kits

- Windows 3.1 SDK
- Windows 3.1 DDK
- Windows 3.0 SDK
- Windows NT (Win32) Pre-Release PDK
- Windows NT (Win32) Pre-Release DDK

Windows 3.1 SDK

The primary Windows development tool you need to do development is the Windows 3.1 Software Development Kit (SDK). It includes the libraries, header files, resource tools, documentation and the Windows debug kernel you need to create native Windows applications.

Before you rush out to buy the SDK, though, note the following points:

- A number of integrated development tools (such as Actor, Visual Basic and Turbo Pascal for Windows) do not require the SDK to operate. See the ***Microsoft Windows Development Tools FAQ*** for details on which tools do not require the SDK.

- A number of compilers (such as Microsoft C/C++ 7.0, Borland C++ 3.0 and Zortech C++ 3.0) include the SDK to operate. See the ***Microsoft Windows Development Tools FAQ*** for details on the extent of the libraries, documentation and resource tools included with various compilers..

The SDK includes the tools you need to create pen-based and multimedia applications, and it also allows you to create applications to run on Windows 3.0.

The list price of the Windows 3.1 SDK is $349.

Windows 3.1 DDK

In order to develop device drivers or VxDs for Windows 3.x, you need to purchase the Windows 3.1 Device Driver Kit (DDK). It includes the necessary header files, libraries, documentation and sample source code to create new device drivers.

The list price of the Windows 3.1 DDK is $495.

Windows 3.0 SDK

The older version of the SDK, 3.0, is still quite useable with Windows 3.1, although it includes older versions of the resource tools, Windows 3.0 debug kernel and is not capable of creating applications which take advantage of the new Windows 3.1 features.

Windows NT (Win32) Pre-Release PDK

The latest membner of the Windows SDK family is the Win32 pre-release Professional Development Kit (PDK). As of July 1992, it is available from Microsoft at the cost of $69 (or $399 for the pre-release PDK plus full printed documentation).

Thie pre-release PDK is supplied on CD-ROM only, and contains the Windows NT operating system as well as all the necessary 32-bit development tools (including a 32-bit C++ compiler). The base package includes all the documentation online on the CD-ROM in PostScript format, while the full package includes printed manuals as well. The purchasers of this PDK will also receive free updates to Windows NT up to and including final release.

Windows NT (Win32) Pre-Release DDK

The Windows NT Device Driver Kit (DDK) will be available from Microsoft in late September for $495. It will include full printed documentation, and be available on CD-ROM only. No base CD-ROM version (without printed documentation) is currently planned.

Programming Techniques

User interface

Finding the number of instances running

Use the following code:

```
        nNumInsts = GetModuleUsage(hInstance);
```

Note that this will always return 1 within Windows NT.

Creating an initially invisible MDI child window

Before creating the child window,

```
SendMessage( hClientWindow, WM_SETREDRAW, 0, 0L);
```

Then, in your child window WM_CREATE processing,

```
ShowWindow( hChildWindow, SW_HIDE).
SendMessage( hClientWindow, WM_SETREDRAW, 1, 0L);
```

Using status bars with MDI **92-09-15**

Add the following code fragments to the indicated places in the `WinProc()` of an application, or the FrameWinProc() of a MDI application.

```
case WM_CREATE:
    hWndClient = CreateWindow( "MDIClient",...,
                    WS_CHILD|WS_VISIBLE|WS_CLIBSIBLINGS|
                    WS_HSCROLL|WS_VSCROLL,... );
    hWndStatus = CreateWindow( "Static",...,
                    WS_CHILD|WS_VISIBLE|SS_LEFT|SS_NOPREFIX,... );
    ...
case WM_SIZE:
    GetClientRect( hWnd,&rect );
    // Calculate DIVIDING_LINE such that.
    // rect.top < DIVIDING_LINE < rect.bottom
    // One choice:
    // DIVIDING_LINE = rect.bottom - GetSystemMetrics(SM_CYMENU);
    MoveWindow( hWndClient,rect.left,rect.top,
                rect.right,DIVIDING_LINE,TRUE );
    MoveWindow( hWndStatus,rect.left,DIVIDING_LINE,
                rect.right,rect.bottom,TRUE );
    break;  // Do *not* pass to DefFrameProc() of MDI app!!!
    ...
    // To change the status text:
    SendMessage( hWndStatus,WM_SETTEXT,0,(LONG)(LPSTR)szStatusText );
```

Notes:

- For non-MDI applications, all references to `hWndClient` above will simply be removed.
- Menu selection can be tracked by setting the status text at a response to the `WM_MENUSELECT` message.
- To draw a line between status bar and the rest of the client area, the `DIVIDING_LINE` should be adjusted in either `MoveWindow()` call to leave a gap between, which is called `InvalidateRect()` for, and actually being painted in response to the `WM_PAINT`.
- The parent window should have `WS_CLIPCHILDREN` style bit set.

Forcing a window to stay fixed size or iconic

In order to make your app stay as an icon, you must process the `WM_QUERYOPEN` message. If you always return 0 for this message, you indicate that the icon can not be opened into a ordinary window.

To retain a fixed size, you must process the `WM_GETMINMAXINFO` message. When you get it, modify the info pointed to by `lParam`:

```
LPPOINT lpSize = (LPPOINT)lParam;
lpSize[3].x = lpSize[4].x = theRightWidth;
lpSize[3].y = lpSize[4].y = theRightHeight;
```

If you don't want the window to be maximized or iconized, create it with the `~WS_MAXIMIZEBOX` and/or `~WS_MINIMIZEBOX` styles, and disable those items from the system menu, if there is one.

Also, you can alternately disable resizing by creating the windows with `~WS_THICKFRAME`, and disabling the `Size...` item on the system menu.

Right-justifying menu items

To right-justify an entire menu item or just a part of it, place a `\a` in the string just before the right-justified part.

Incidentally, the Windows 3.0 CUA guidelines no longer call for right-justifying the Help menu on the menu bar.

Right-justifying menu items at runtime

It's undocumented, but what you need is a `0x08` in the menu string.   The easiest way to do this is to place a `\b` in the string before the right-justified part (either the text of the accelerator key).

Incidentally, the Windows 3.0 CUA guidelines no longer call for rightjustifying the Help menu on the menu bar.

Drag-and-drop: File Manager and Print Manager

You will need to register your application in the registration database. You can do this either using the Registration Editor, or the `Reg*` API in Windows 3.1 SDK. One of the simplest mechanisms is that used by several Windows 3.1 applets to print a file the parameters are

```
/p filename
```

See the registration database for examples.

Drag-and-drop: generalized

To do generalized drag-and-drop, you'll need SHELL.DLL, shipped with Windows 3.1.

Either do `DragAccept()` or create the window as `WS_EX_DROPFILES (0x10L)`

Wait for the `WM_DROPFILES` message `(0x0233)`, which passes a handle to something in wParam

You can then issue

```
WORD DragQueryFile(hDrop, 0xffff, NULL, 0)
```

to get the file count, and then

```
WORD DragQueryFile(HANDLE hDrop, WORD nFile, LPSTR sDest, WORD max)
```

for each of the dropped files

Once you have finished, call

```
DragFinish( hDrop )
```

For Visual Basic, get the file DD.ZIP from *CompuServe*'s MBASIC forum, which lets you implement drag-and-drop from VB. This file may also be available at *ftp.cica.indiana.edu.*

Minimize button on modal dialog moves when clicked

It's a bug in Windows 3.1. To duplicate this, create a modal dialog with the styles `CAPTION, MODAL FRAME, MINIMIZE-BOX`, activate the dialog, press the Minimize button -- and watch it move to the top right corner, on top of the modal frame!

The workaround: don't use a Minimize box on a modal dialog.

Trapping mouse clicks on desktop

To trap mouseclicks on the desktop, you will need to subclass the desktop window. The following code fragment illustrates the technique (sample code courtesy of Blake Coverett, blakeco@microsoft.com):

To subclass the desktop:

```
hWndDesktop=GetDesktopWindow();
hSaveTask=GetCurrentTask();
lpfnDesktop=(FARPROC)GetWindowLong(hWndDesktop, GWL_WNDPROC);
lpfnSubClassProc=MakeProcInstance((FARPROC)WndProc, hInst);
SetWindowLong(hWndDesktop, GWL_WNDPROC,
              (LPARAM)(LONG)lpfnSubClassProc);
```

and then to undo it when ready to unload:

```
SetWindowLong(hWnd,GWL_WNDPROC, (LPARAM)(LONG)lpfnDesktop);
PostAppMessage(hSaveTask,WM_QUIT,0,0);
```

Dialogs and controls

Hiding dialog controls

```
EnableWindow(GetDlgItem(hDlg, IDD_CONTROLTOHIDE), FALSE);
ShowWindow(GetDlgItem(hDlg, IDD_CONTROLTOHIDE), SW_HIDE);
UpdateWindow(GetDlgItem(hDlg, IDD_CONTROLTOHIDE));
```

Adding controls to a non-dialog window

You can do this by simply calling `CreateWindow()` with one of the predefined child window control class names (see the control class definition table in the SDK reference manual).

Preventing switching away from a modal dialog

The design of the Windows API means that if there are two dialogs active simultaneously, the user can switch between the two, even one of them is modal. To prevent this, you should use `EnableWindow()` to explicitly diable any modeless dialogs when your modal dialog starts up.

Subclassing standard controls

You can *subclass* standard controls by having your own window procedure handle the messages for the windows you create (using `SetWindowLong()`). The only caveat here is for useability: make sure that your subclassed controls don't behave in an unexpected manner.

What is definitely a bad idea is modify the *class* procedure of a standard control (using `SetClassLong()`) and changing the window procedure for *all* such windows, as this will affect all edit controls in all applications currently running in the Windows session.

Allowing ENTER in a multiline edit control **92-09-15**

To allow the use of the **Enter** key, there is no need to subclass the edit control. An easier way (which also works better!) is to specify `ES_WANTRETURN` as part of the style for the edit control (see the Windows 3.1 SDK documentation for details).

Aligning multi-column listboxes **92-09-15**

In the resource file make sure the list box has the `LBS_USETABSTOPS` style. When you add the items to the listbox, separate the fields with tabs. You can either use the default tab stops, or set your own by sending the `LBS_SETTABSTOPS` message to the listbox. For more information, see the SDK Reference, volumes 1 and 2.

It is also possible to use a fixed font, but the tabstop solution usually ends up looking much better.

Changing button colors

In Windows 3.0, the button face is defined by two colors. The grey (white with EGA) face and a dark grey (grey if ega) shadow. The colors also change when the button goes from a normal to pushed in state. The `WM_CTLCOLOR` message only allows you to change one color at a time so to which of the button face colors should this apply? (Windows 2.x button faces had only one color so it made sense.)

Maybe something tricky could have been done by using the background color for the shadow and foreground color for the face and perhaps doing something strange to get the text color in another way... And how do you return 2 brushes (you now need a foreground and a background brush)? Or maybe even better, make colors a property of the window and some windows could have multiple color properties...

Anyway, Windows doesn't look at the `WM_CTLCOLOR` message for buttons and thus doesn't allow you to change the button colors. Try it with a listbox instead... The only way to change button colors is to specify *ButtonColor*=, *ButtonShadow*= and *ButtonText*= in the [Colors] section of your **win.ini** file.

In Windows 3.1, the button text, shadow and face colors can also be defined using the Control Panel.

Doing a timeout in a dialog

Start a timer in your `WM_INITDIALOG` processing. If your dialog box receives the `WM_TIMER` message, kill the timer and post yourself a `WM_COMMAND` messgae with `wParam == IDOK`. If the user presses any button, restart the timer.

Null dialog handles from Borland custom dialogs

If you keep getting null dialog handles with Borland C++ unless I have Turbo C++ running, your dialog is probably of the `BorDlg` class, which requires code in `BWCC.DLL`. However, you have probably not done anything to force `BWCC.DLL` to be loaded with your application, so the dialog manager cannot find the necessary routines to draw the dialog. The easiest way to force `BWCC.DLL` to be loaded is to call `BWCCGetVersion()` at the very beginning of your application, and to link in `BWCC.LIB`.

Alternatly, you can do a

```
WinExec("loadbwcc.exe");
```

when you start up your application, as long as `loadbwcc.exe` is available.

Memory

Using new() in C++

In Borland C++ 2.0, and in 3.x's medium model, new() ends up calling `LocalAlloc()`, allocating memory from your near 64K segment. In BCC 3.x's large and compact models (and in Microsoft C/C++ 7.0), however, it will make one `GlobalAlloc()` and do subsegment allocations to allow you access to the full memory without making excessive demands on the system limit of 4096 (8192 in 386 enhanced mode) global memory handles.

Global memory owned by DLL

If you use GlobalAlloc in a DLL, the application that called the DLL will own the object. There is a way around this, though: allocate the memory using the `GMEM_DDESHARE` flag; this will make the allocating code segment (rather than the current task) own the memory.

Determining size of physical memory

You need to make a DPMI call to obtain that piece of information. DPMI call `0500h` with `ES:DI` pointing to a `30h` byte buffer returns the "Free Memory Information":

Offset	**Description**
`00h`	Largest available free block in bytes
`04h`	Maximum unlocked page allocation
`08h`	Maximum locked page allocation
`0Ch`	Linear address space size in pages
`10h`	Total number of unlocked pages
`14h`	Number of free pages
`18h`	Total number of physical pages
`1Ch`	Free linear address space in pages
`20h`	Size of paging file/partition in pages
`24h-2Fh`	Reserved

The size of one page in bytes can be determined by function `0604h`, which returns the page size in bytes in `BX:CX`. To call a DPMI function, invoke the interrupt `31h`. Carry bit will be clear if call was successful.

The complete DPMI 0.9 specification is available free from Intel Literature JP26, Santa Clara. It's also available on ftp.cica.indiana.edu.

GDI

Background color

If you insist on a white background, use

```
WinClass.hbrBackground = GetStockObject(GCW_WHITEBRUSH);
```

for your window background. If it doesn't matter to you, however, you should use the *Control Panel*-defined window background color instead:

```
WinClass.hbrBackground = CreateSolidBrush(COLOR_WINDOW + 1);
```

Changing palette entries in 16-color mode

If you are using a standard driver, you will need to bypass Windows to do it (if you happen to have a 16-color driver which support palettes, you can use standard Windows palette management functions).

Microsoft will tell you to buy the DDK, but there is another way. Now, the Windows system palette maps onto the VGA 16-color palette as follows:

VGAPAL	SYSPAL	VGAPAL	SYSPAL
00	00	08	07
01	01	09	13
02	02	10	14
03	03	11	15
04	04	12	16
05	05	13	17
06	06	14	18
07	12	15	19

So you can define some macros to take care of the mapping:

```
#define syspal(n) (n<7 ? n : (n>8 ? n+4 : (n=7 ? 12 : 7)))
#define vgapal(n) (n<7 ? n : (n>12 ? n-4 : (n=7 ? 8 : 7)))
```

When you get a WM_SETFOCUS event, save the current state of the hardware colormap and installs the one you want. When you get a WM_KILLFOCUS event, restore the original palette. Don't use the palette registers directly, though, just modify the color registersthat they point to. (For details on redefining a VGA palette, see a book such as *A Programmer's Guide to PC and PS/2 Video Systems* by Richard Wilton.)

Miscellaneous

Wsprintf and sprintf

`wsprintf()` can not print floating-point numbers by design. To print floating-point, you must use `sprintf()`. Remember, though, that all strings passed to `wsprintf()` should be cast to FAR!

Changing your current directory

The easy way is to use `DlgDirList()`. You can specify zero for the two ID fields. You can use the current window ID for the dialog handle field.

The standard C library functions `chdir()` and `getcwd()` can also be used.

Detecting idle time 92-09-15

The 'idle-detecting' message loop may suit your case. That is, replace the standard `while( GetMessage() )` ... in `WinMain()` with the something like the following (thanks to Risto Lankinen for the example):

```
if ( PeekMessage(&msg) ) {
    // The queue contains messages - process them
    // GetMessage() would automatically detect WM_QUIT, but
    // we must explicitly check for it.
    if ( msg.message == WM_QUIT )
        break;
    if ( TranslateAccelerator(hWnd,hAcc,&msg) )
        continue;

    TranslateMessage( &msg );
    DispatchMessage( &msg );

    // You might want to save the last time a message
    // was processed
    dwLastMsgTime = GetTickCount();
} else {
    // The queue is empty - user is doing nothing with *this* app
    if ( GetTickCount() < dwLastMsgTime ) {
        // Timer wrapped around -- do something!
    } else if ( GetTickCount() - dwLastMsgTime > MSGTIMEDELTA ) {
        // Do something funky
    }
```

Extracting icons from a .EXE or .DLL

In Windows 3.1, it's easy to enumerate the icons in a Windows EXE or DLL even if you don't already know their names. SHELL.DLL exports

```
HICON ExtractIcon(hInst, lpszExeName, nIcon)
```

This function returns a handle to the specified icon (where 0 is the default icon displayed by Program Manager), or the number of icons in the file if you pass in an index of -1.

Better yet, SHELL.DLL also exports the function:

```
FindExecutable(lpszFile, lpszDir, lpszResult)
```

which will give you the executable associated with a given document file. You can then extract the appropriate icon from that file.

Multimedia RIFF file format **92-09-15**

The IBM/Microsoft RIFF and MCI definition document is available for anonymous ftp download from the */vendor/microsoft/multimedia* directory on *ftp.uu.net*. If you don't have anonymous ftp access, try CompuServe in the WINSDK forum, or Microsoft Online.

The document describes multimedia interfaces (MCI) and data formats (RIFF). IBM has committed to include support for these in OS/2. These interfaces are already supported in the Multimedia Extensions for Windows (MME) and Windows 3.1. Included in the RIFF file format is a waveform (.WAV) audio definition; this format is the system standard for Windows and OS/2.

Interfacing to the outside world

Passing a pointer to a DOS application or TSR 92-09-15

In order to pass a pointer to a DOS application (to share memory), you can **not** just pass a Windows pointer. `GlobalLock()` returns a segment selector table entry, not a physical address. Thus, the simple code below will give you an incorrect address:

```
lpBuffer    = GlobalLock(hBuffer);
InRegs.x.di = FP_OFF(lpBuffer);
SegRegs.es  = FP_SEG(lpBuffer);
int86x(0x7f,&InRegs,&OutRegs,&SegRegs);
```

The problem is that the TSR or DOS application runs in real mode, while Windows applications running in Standard or 386 Enhanced mode use selectors [LDT] and not pointers[SEG:OFF] to access memory.

The following gives an outline of what needs to be done, courtesy of Glenn Boozer (glenn@imagen.com):

To send a pointer to DOS [a Segment:Offset address, not a protected mode selector]

- DosAllocate a memory buffer [This will be a buffer in the first 640K of address space. This buffer is **locked** and will not move. *[Not recomended by Microsoft]*
- Copy the data from the buffer that is in "Windows Application space" into the DOS Buffer
- Get the Segment and Offset of the DOS buffer and pass that to the TSR.
- Release the DOS Buffer

To use a pointer [Segment:offset] you got from a DOS application:

- Allocate a Selector *[Not recomended by Microsoft]*
- Set the selector base and length with the data returned from the TSR. *[Not recomended by Microsoft]*
- Use the data
- Release the selector. *[Not recomended by Microsoft]*

Selected code fragments follow.

```
// Windows kernel calls not in WINDOWS.H
WORD  FAR PASCAL SetSelectorBase(HANDLE hSelector, DWORD dwBase);
WORD  FAR PASCAL SetSelectorLimit(HANDLE hSelector, DWORD dwLimit);
DWORD FAR PASCAL GlobalDosAlloc(DWORD);
WORD  FAR PASCAL GlobalDosFree(WORD);

HANDLE FAR PASCAL
GetPhysicalMemoryHandle()
{
    HANDLE hSel;
    HANDLE hSel2;

    /*. create a selector for use by MakePhysicalMemoryPtr() */
    /* The how of this is taken from an SR response. */
    if ((hSel2 = GlobalAlloc(GMEM_FIXED,(long) 64)) != NULL) {
        hSel = AllocSelector(hSel2);
        GlobalFree(hSel2);
    } else {
        hSel = (HANDLE)NULL;
    }
    return hSel;
}
```

```
LPSTR NEAR PASCAL
MakePhysicalMemoryPtr(WORD wMemHandle, WORD wSegment, WORD wOffset)
{
    /*. set selector base from wSegment parameter */
    SetSelectorBase(wMemHandle, (((LONG)wSegment)<<4) + wOffset );
    /*. set limit for 4K bytes accessable */
    SetSelectorLimit(wMemHandle,(long) 0x0FFF);
    /*. make and return a long pointer using passed wMemHandle */
    return (LPSTR)MAKELONG(0, wMemHandle);
}

void FAR PASCAL
FreePhysicalMemoryPtr(LPSTR lpMemPtr)
{
    FreeSelector(HIWORD(lpMemPtr));
}
```

Main code fragment:

```
// Assuming protected mode
if (!( hPhysMemHandle = GetPhysicalMemoryHandle())) {
    MessageBoxOKHand((LPSTR) "Error: Could not get "
                             "physical memory for TSR");
    return;
}

// [call tsr-calling routine n times]

// Return the handle we allocated
FreePhysicalMemoryPtr(lpPhysPtr);
```

TSR-calling routine:

```
static FPTR   near fp;

lpsDosParagraphSelector.d =
              GlobalDosAlloc((DWORD)max( APImsg.len, 4));
if(APImsg.buffer) {
    lmemcpy( (LPSTR)MAKELONG(0, lpsDosParagraphSelector.w.sel),
                             APImsg.buffer, max( APImsg.len, 4));
    fp.w.sel = lpsDosParagraphSelector.w.par;
    fp.w.off = 0;
} else {
    /* Null Pointer */
    fp.p = 0L;
}
dx = fp.w.sel;
bx = fp.w.off;

// Call the TSR
rc = int2f(ax, cx, si, di, dx, bx, (unsigned int far *)&di,
          (unsigned int far *)&si, (unsigned int far *)&cx,
          (unsigned int far *)&dx, (unsigned int far *)&bx);

(void) GlobalDosFree(lpsDosParagraphSelector.w.sel);
lpPhysPtr = MakePhysicalMemoryPtr(hPhysMemHandle, dx, bx);
fp.p = lpPhysPtr;
```

Starting a Windows application from a DOS session 92-09-15

This is really quite difficult, and you may be happier using an existing implementation (such as *wx* and *wxserver,* which comes with the Windows 3.1 SDK), because in 386Enhanced mode the DOS application and the Windows world are in separate virtual machines; the only context they have in common is the underlying DOS. The basic idea is to use a TSR that talks to both the DOS app and a Windows "wrapper" app that does the `WinExec()` for you. Thanks for the explanation are due to Ed Schwalenberg (ed@odi.com).

Create a TSR that gets loaded before Windows is started. Its services will be available to both DOS apps and Windows apps. When Windows is started, your wrapper program can call the TSR with an `INT 2F`, giving it the address of some `GlobalDOSAlloc()`'ed memory which will be used to pass information back and forth between the TSR and protected-mode Windows. While processing this `INT 2F`, you issue one of your own, with `AX=1683h`, which will return in `BX` the magic number of the Windows virtual machine (VxD), which is currently 1 but may change.

The DOS application issues an `INT 2F`, passing the name of the desired application to the TSR. The TSR copies the information into a private data buffer in the TSR's address space, NOT to the `GlobalDOSAlloc()`'ed memory (which only exists in the Windows virtual machine).

Now for the hard part. You need to call back to Windows when the Windows VM is scheduled. To do that, use `INT 2F`, `AX=1685h`, `BX=`Windows VM number which you saved from the initialization step, `CX=`flags, `DS:SI=` priority boost and `ES:DI=CS:IP` of a procedure to call. When the Windows VM is scheduled, your procedure will be called. That procedure can copy the name of the application into the `GlobalDOSAlloc()`'ed memory, issue an `INT 2F` to the windows wrapper program, and `IRET`. The windows wrapper program can use the data in the `GlobalDOSAlloc()`'ed memory to `WinExec()` the desired program.

Mixed-language programming

Visual Basic and Fortran

If you want to use Visual Basic to build a slick interface for your old text-based Fortran code, the approach to take is to build the Fortran code into a DLL, and call it from Visual Basic. You may either pass the parameters as arguments, or you may want to construct a temporary file for more extensive input.

Passing a structure back to Visual Basic from a DLL

The following description is courtesy of Todd Ogasawara, 1991 (reachable at todd@pinhead.pegasus.com). The code fragments were developed and tested using Borland C++ 2.0 and Microsoft Visual Basic.

Define a type that is a pointer to a structure.

```
typedef struct {
   long    fsize;          // file size in bytes
   char    ftime[25];      // last file access time string
} * fileStruct;
```

Sample function prototype declaration.

```
int FAR pascal FileInfo(char filename[], fileStruct);
```

Sample DLL C function that receives a filename in a char array from Visual Basic and passes back file size (`long`) and file modification date (`char` array) in a structure.

```
// Get file info (access time & size)
int FAR pascal
FileInfo(char filename[], fileStruct far fileinfo)
{
   struct stat statbuf;
   FILE   *stream;
   if (!(stream = fopen(filename, "r"))) {
         return(-1); // ERROR: cannot find named file
   } else {
         fstat(fileno(stream), &statbuf);
         fclose(stream);
   }
   /* file size */
   fileinfo->fsize = statbuf.st_size;
   /* access time */
   strcpy(fileinfo->ftime,ctime(&statbuf.st_ctime));
   return(0);
}
```

Declare a Visual BASIC "user-defined type" (i.e., a "structure") that matches the structure declared in the DLL C code. See pages 260-261 of the Microsoft Visual BASIC Programmer's Guide for more information about user-defined types.

```
' type (structure) definition in GLOBAL.BAS
Type FileStruct
   Fsize As Long
   Ftime As String * 25
End Type
```

Declare the function in your `GLOBAL.BAS` (or whatever you named the file you keep global information in). In this example a function is declared since the DLL C function returns a -1 to indicate an error and a 0 to indicate success. Note that the filename is passed from Visual BASIC to the DLL C function by value (`ByVal`) while the data in the DLL C structure is passed to Visual BASIC by reference (`As`). See pages 379-387 of the Microsoft Visual BASIC Programmer's Guide for more information about declaring and calling DLL routines.

```
' declaration in GLOBAL.BAS
Declare Function FileInfo Lib "dosdll.dll" (ByVal FileName$,
   FileInf As FileStruct) As Integer
```

Example of using the DLL function 'FileInfo' in Visual BASIC.

```
If (Myfile.Filename = "") Then
   Exit Sub
Else
   ThisFileName$ = UCase$(Myfile.Filename)
End If
FileStatus% = FileInfo(ThisFileName$, FileStat)
ThisFileSize$ = Format$(FileStat.Fsize, "###,###,###") + "
   bytes" ThisFileStat$ = Left$(FileStat.Ftime, 24)
```

Putting it all together

Compiling and linking

Large memory model: why or why not?

Yes, you can do it. There are several problems with using large model, though:

> Your program's data memory will be fixed in real mode. Effectively, your application will cripple any real-mode Windows system. (Of course, this problem doesn't exist with Windows 3.1!)
>
> Your application will run more slowly, since all your data must be accessed through far pointers.
>
> You may only be able to have one instance of your application active at any one time. This restriction is imposed by Windows on applications with multiple static data segements: in large model, that means most applications generated with C or C++. Borland C++ 3.0+ and Microsoft C/C++ 7.0 will attempt to keep your static and global data in a single segment (although Microsof C5.1 an C6.0 will not), so as long as that data does not exceed 64K, you could run multiple instances of a large-model application created using those compilers.

You should consider very carefully before you decide that large model is the only way to go; the preferred method is to use medium model, and to allocate far data as required.

Another alternative is to use a compiler such as Watcom C/386 or Zortech C++ for development; this will let you use a single 4GB segment, and 32-bit registers, increasing your applications performace substantially (but limiting it to running in 386 enhanced mode).

Finally, developing for Win32 (which encompasses both the upcoming Windows NT and Win32s) will allow you to use the flat 32-bit memory model without the restrictions and penalties associated with using a 32-bit environment on Windows 3.x.

Emulator vs. alternate floating-point math

The alternate math package is faster on non-x87 machines, but slower on those equipped with a math chip. Depending on your application, you might want to ship either, or both. If you need accuracy in floating-point calculations, though, stay away from the alternate math package.

Borland C++ does not support the alternate math package, but it does have a "fast floats" option, which is roughly equivalent.

Emulator floating-point: corrupted code segments

Compiling a Windows application with emulator floating-point causes corrupted code segments when running on a non-8087 equipped system in Windows 2.x and Windows 3.0 Real mode.

The emulated floating point tries to used the coprocessor. When it does not find one on startup, it patches the code to use the software floating point. Patching does not, however, recalculate the code-segment checksum, thus the Windows debugging kernel chokes when it finds that something terrible must have happened to the code. (This problem does not affect Windows 3.x in Standard or 386 Enhanced modes.)

You can get Windows to ignore the checksum errors by setting *EnableSegmentChecksum=0* in the *[debug]* section of **win.ini**; the problem only affects debugging versions of Windows 3.0.

Debugging

Turbo Debugger video configuration **92-09-14**

Borland's TDW debugger uses its own video drivers for single-screen Windows debugging. As a result, you must make sure that the debugger can find the correct drivers. As an example, if you are using an ATI Graphics Ultra (or Graphics Vantage), you need to make sure the following lines are in your **tdw.ini** file:

```
[Debugger]
VideoDLL=C:\TPW15\ULTRA.DLL

[VideoOptions]
DebugFile=C:\TDW.LOG
```

Dr. Watson log files

The Dr. Watson log shows the contents of the registers when you application crashed. Even if you can't use it to determine contents of the variables, you can pinpoint the location of the crash.

Make sure you keep a copy of the `.map` file generated by the linker for the version shipped to your customers; you can then look up the crash location manually from this file when you receive a Dr. Watson log. If you linked a version with `/CO /LI`, the .map file will also contain line number information, allowing you to pinpoint the line in your program.

Resources and tools

Tracking down unfreed resources

There are several utilities available on *ftp.cica.indiana.edu* which will monitor the heap and memory usage. Look for files `ma.zip` and `ha.zip` in */pub/pc/win3/utils.* The Windows 3.1 debug kernel (included with the SDK) also checks for unfreed resources when an application exits.

Linking fonts into a .FON file

The linker provided with the Windows 3.0 SDK will produce the following error when linking fonts:

```
Link Error L2049: no segments defined
```

The above LINK error is a bug in link. The fix is to run `exehdr /r` on the `.exe` file, and then run `rc` on it. The Windows 3.0 SDK linker incorrectly detects an error, and marks the resulting .exe file with some kind of error bit, even though the rest of the exe file is ok. `Exehdr /r` will reset this "error bit", after which rc will work just fine.

An alternate fix is to use `link4` from Windows 2.x SDK.

Documentation and help

Adding bitmaps to helpfiles

When you add a bitmap to a help source document using the `[bml ...]` command, it freqently does not appear in the compiled helpfile. The problem is that the text `[bml printer.bmp]` is an RTF bitmap inclusion command (which is why you want it there), but Word assumes you really want the literal text "`[bml printer.bmp]`", and escapes the whole sequence when saving the files as RTF. *Note that the previous description substitutes square brackets for curly brackets to prevent* hc *from actually including those bitmaps. Use curly brackets in your own helpfile source!*

You'll get the actual RTF bitmap inclusion command in the RTF file (and thus a bitmap in the compiled helpfile) by inserting a bitmap using the Word for Windows menu commands and clicking on the "Link to file" checkbox when it asks you which bitmap to insert.

Screen Snapshots

To take a snapshot of your screen, just press **PrtScr**, and Windows will copy the image to the clipboard, from where you can paste it into your favourite application. You can also use **Alt+PrtScr** to take a snapshot of only your active pop-up window (child windows such as dialog controls are not counted as "active").

You may wish to select the Monochrome VGA driver prior to doing the screen print to produce 1-bit (2-color) bitmaps for easier printing.

As an alternative, you may wish to use a screen grabber/graphics conversion utility such as HiJaak or PaintShop Pro (see the ***Microsoft Windows FAQ***) if you wish to produce good-quality grayscale bitmaps..

A programmer's bibliography

Windows 3.1 SDK

Windows Programmer's Reference, Volume 1: Overview. Microsoft Press, 1992

Windows Programmer's Reference, Volume 2: Functions. Microsoft Press, 1992

Windows Programmer's Reference, Volume 3: Messages and structures. Microsoft Press, 1992

Windows Programmer's Reference, Volume 4: Resources. Microsoft Press, 1992

Windows Guide to Programming. Microsoft Press, 1992

Windows Programming Tools. Microsoft Press, 1992

Windows User Interface Guidelines. Microsoft Press, 1992

Windows Multimedia Programmer's. Guide Microsoft Press, 1992

Windows Multimedia Programmer's Reference. Microsoft Press, 1992

Windows for Pen Computing Programmer's Reference. Microsoft Press, 1992

Windows Setup Toolkit. Microsoft Press, 1992

Windows 3.0 SDK

SDK Reference, Volume 1: Functions and messages. Microsoft Press, 1990, part no. 06856

SDK Reference, Volume 2: Resource scripts and file formats. Microsoft Press, 1990, part no. 06857

SDK Guide to Programming. Microsoft Press. 1990, part no. 06854

SDK Tools. Microsoft Press, 1990, part no. 06854

SAA CUA Advanced Interface Design Guide. IBM, 1989, part no. SC26-4582-0

Win32 (Windows NT) API

Win32 API: An Overview. Microsoft Press, 1992

Win32 Programmer's Reference, Volume 1: Overview, functions (A-O). Microsoft Press, 1992

Win32 Programmer's Reference, Volume 2: Functions (P-Z). Microsoft Press, 1992

User interface guidelines

Windows User Interface Guidelines. [for Windows 3.1, Pen Windows and Windows NT] Microsoft Press, 1992

SAA CUA Advanced Interface Design Guide. [for Windows 3.0 and OS/2 1.x] IBM, 1989, part no. SC26-4582-0

SAA CUA'91 Design Guide. [for OS/2 2.0] IBM, 1991, part no. SC34-4289, $10.00

SAA CUA'91 Reference. [for OS/2 2.0] IBM, 1991, part no. SC34-4290, $18.25

Third-party programming guides

Charles Petzold: *Programming Windows*, 2nd edition. Microsoft Press, 1990, ISBN 1-55615-264-7

Jeffrey M. Richter: *Windows 3: A Developer's Guide.* M&T Books, 1991, ISBN 1-55851-164-4

Daniel A. Norton: *Writing Windows Device Drivers.* Addison-Wesley, 1991, $29.95

Peter Norton and Paul Yao: *Windows 3.0 Power Programming Techniques.* Bantam Books, 1991, $29.95

Richard Wilton: *Microsoft Windows 3 Developer's Workshop.* Microsoft Press, 1991, $24.95